LINUX BASICS

Your Journey Starts Here

FREE & OPEN SOURCE | SECURE & STABLE | MULTIUSER SYSTEM | HIGHLY CUSTOMIZABLE

OPEN SOURCE
Linux is free to use, modify and distribute.

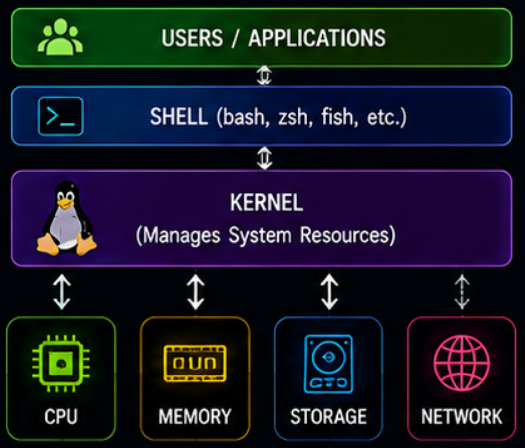
KERNEL
The core of the OS that manages hardware and system resources.

TERMINAL
A powerful text interface to interact with the system.

SHELL
A command-line interpreter that acts as a bridge between you and the kernel.

DISTRIBUTIONS
Linux comes in many flavors called distributions (distros).

HOW LINUX WORKS?



TERMINAL EXAMPLE

```
user@linux: ~
user@linux:~$ pwd
/home/user
user@linux:~$ ls -l
total 4
drwxr-xr-x  2 user user 4096 May 20 10:30 Documents
drwxr-xr-x  2 user user 4096 May 20 10:30 Downloads
-rw-r--r--  1 user user  220 May 20 10:30 notes.txt

user@linux:~$ whoami
user
user@linux:~$
```

The terminal is your best friend in Linux!
Learn commands, automate tasks, and boost productivity.

POPULAR LINUX DISTRIBUTIONS

UBUNTU — User-friendly and perfect for beginners.

DEBIAN — Stable, secure and the base for many distros.

FEDORA — Latest features and cutting-edge technology.

OPENSUSE — Great for both beginners and sysadmins.

ARCH LINUX — Minimal, flexible and built for advanced users.

BASIC LINUX COMMANDS

ls	List files and directories
cd [dir]	Change directory
cat [file]	View file content
cp [src] [dest]	Copy files or directories
rm [file]	Remove files or directories
mv [src] [dest]	Move or rename files
grep 'text' [file]	Search text in files
man [command]	Show manual of a command

Tip: Most commands have options. Try: man <command> to learn more!

LINUX FILE SYSTEM STRUCTURE

/	Root directory
/bin	Essential user binaries
/etc	System configuration files
/home	User home directories
/usr	User programs & data
/var	Variable data (logs, caches)
/tmp	Temporary files
/boot	Boot loader files
/dev	Device files

Everything in Linux is a file!

WHY USE LINUX?

Secure	Built with security in mind.
Stable	Runs for months or years without reboot.
Free	No license cost. 100% Open Source.
Powerful	Perfect for developers, servers & more.
Community	Huge community support worldwide.

Learn | Practice | Explore | Automate

Linux is not just an OS, it's a way of thinking!

Open Source, Open Mind

LINUX DISTRIBUTIONS

Popular Linux Distros for Every User

Linux™

Download | Install | Try | Code | Terminal | Explore

1. UBUNTU

ubuntu

Ubuntu is one of the most popular Linux distributions. It is user-friendly, stable and perfect for beginners.

- Easy to use
- Large community support
- Regular updates
- Great for desktop & server

Best For: Beginners, Desktop Users, Developers

2. DEBIAN

debian

Debian is known for its stability, security and huge software repository. It is the base for many distros.

- Very stable
- Huge package repository
- Strong security
- Free and open source

Best For: Servers, Advanced Users, Stable Systems

3. FEDORA

fedora

Fedora is a modern, fast and innovative distro. It includes the latest software and technologies.

- Latest software
- Secure and reliable
- Great for developers
- Community driven

Best For: Developers, Tech Enthusiasts, Testing New Features

4. ARCH LINUX

archlinux

Arch Linux is for advanced users who want full control and a minimal base system.

- Minimal and lightweight
- Full customization
- Rolling release
- Great documentation (Wiki)

Best For: Advanced Users, Power Users, Customization

5. LINUX MINT

linuxmint

Linux Mint is based on Ubuntu and is famous for its clean interface and ease of use.

- User friendly
- Out of the box experience
- Stable and reliable
- Great for daily use

Best For: Beginners, Everyday Users, Home & Office

6. KALI LINUX

KALI LINUX

Kali Linux is designed for penetration testing, ethical hacking and cyber security professionals.

- Pre-installed security tools
- Pen testing & forensics
- Regular updates
- Advanced security distro

Best For: Ethical Hackers, Security Experts, Penetration Testing

7. CENTOS

CentOS

CentOS is a stable and reliable enterprise-class distro. It is based on RHEL (Red Hat Enterprise Linux).

- Enterprise grade
- Very stable
- Long term support
- Great for servers

Best For: Servers, Enterprises, Cloud Environments

8. OPENSUSE

openSUSE

openSUSE is a versatile distro with powerful tools for system administration and development.

- Easy installation
- Great admin tools
- Stable and flexible
- Two editions: Leap & Tumbleweed

Best For: Developers, System Admins, Advanced Users

QUICK TIP

Choose the right distro based on your needs.
All Linux distributions are free and open source!

Terminal | Open Source | Secure | Highly Customizable | Strong Community

Download Official ISOs from the official websites only to stay safe & secure.

TERMINAL BASICS

Understand the Building Blocks of Linux Terminal

Learn | Practice | Use | Automate | Execute | Master

TERMINAL

The Terminal is a program that allows you to interact with the Linux system using text commands.

SHELL

The Shell is a command interpreter that takes your commands and gives them to the operating system.

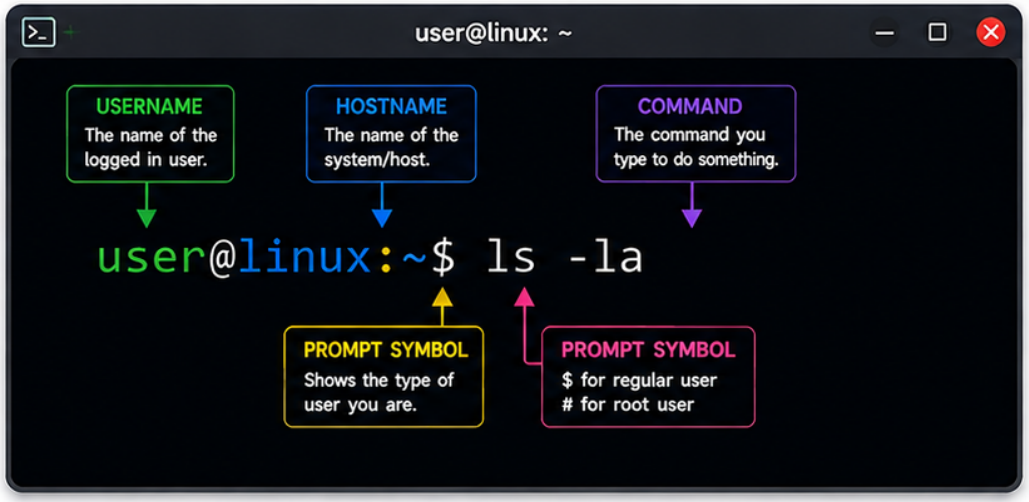
COMMAND

A command is an instruction you give to the system to perform a specific task.

PROMPT

The prompt is the symbol that shows the terminal is ready for you to type a command.

TERMINAL ANATOMY



USERNAME

The name of the current user who is logged in and running the terminal.

HOSTNAME

The name of the machine or system you are working on.

$ /

$ → Regular user

→ Root (Administrator) user

CLI

Command Line Interface lets you control your system by typing commands.

WHAT IS CLI?

CLI (Command Line Interface) allows you to interact with your system by typing text commands instead of using a graphical interface.

Why CLI?

- ✓ Faster
- ✓ Uses less resources
- ✓ Powerful for automation
- ✓ Works on any system

TYPES OF PROMPT

Regular User Prompt ($)

```
user@linux:~$
```

You are a regular user (limited permissions)

Root User Prompt (#)

```
root@linux:~#
```

You are the root user (full system access)

COMMON PROMPT EXAMPLES

`user@linux:~$`	In home directory of regular user
`user@linux:/home/user$`	In a specific directory
`user@linux:/var/log$`	In system directory
`root@linux:~#`	Root user in home directory
`root@linux:/etc#`	Root user in /etc directory

BASIC COMMANDS TO TRY

Command	Description
`whoami`	Show current username
`hostname`	Show system hostname
`pwd`	Print working directory
`ls`	List files and directories
`cd <dir>`	Change directory
`clear`	Clear the terminal screen
`exit`	Exit the terminal

QUICK TIP

Practice regularly. The more you use the terminal, the more powerful you become! 🚀

Practice | Learn | Secure | Automate | Community | Progress

Use the terminal daily and become a Linux Pro! 🐧

Linux™ NAVIGATION COMMANDS

Move Around the Linux File System Like a Pro

Learn | Practice | Use | Script | History | Master

LINUX FILE SYSTEM STRUCTURE

Everything in Linux starts from the root directory (/).

- /
  - bin — Essential user binaries
  - boot — Boot loader files
  - dev — Device files
  - etc — System configuration
  - home — User home directories
  - lib — System libraries
  - opt — Optional applications
  - tmp — Temporary files
  - usr — User programs & data
  - var — Variable files (logs, etc.)

TIP

Use these commands to navigate, explore and manage directories efficiently.

1. pwd

Print Working Directory

Shows the full path of the current working directory.

Syntax

```
pwd
```

Example

```
user@linux:~$ pwd
/home/user
```

Tells you where you are.

2. ls

List Directory Contents

Lists files and directories in the current directory.

Syntax

```
ls [options] [directory]
```

Example

```
user@linux:~$ ls
Desktop   Documents  Music
Pictures  Videos     file.txt
```

Use ls -la for detailed list.

3. cd

Change Directory

Changes the current directory to the specified location.

Syntax

```
cd [directory]
```

Example

```
user@linux:~$ cd Documents
user@linux:~/Documents$
```

Use cd .. to move up one level.

4. tree

Show Directory Tree

Displays the directory structure in a tree format.

Syntax

```
tree [directory]
```

Example

```
user@linux:~$ tree
.
├── Desktop
├── Documents
│   └── file.txt
└── Pictures
2 directories, 1 file
```

May need to install: sudo apt install tree

5. clear

Clear Terminal Screen

Clears the terminal screen for a fresh start.

Syntax

```
clear
```

Example

```
user@linux:~$ clear
```

Keeps your terminal clean.

6. history

Command History

Shows the list of previously executed commands.

Syntax

```
history
```

Example

```
user@linux:~$ history
  1  ls -la
  2  cd Documents
  3  pwd
  4  clear
...
```

Use !n to run command number n.

7. dirs

Show Directory Stack

Displays the list of directories in the stack.

Syntax

```
dirs
```

Example

```
user@linux:~$ dirs
0 /home/user/Documents
1 /home/user
```

Shows saved directories stack.

8. pushd

Push Directory to Stack

Saves current directory and changes to the new directory.

Syntax

```
pushd [directory]
```

Example

```
user@linux:~$ pushd /var/log
~/var/log ~/Documents ~
```

Directory is added to stack.

9. popd

Pop Directory from Stack

Returns to the previous directory in the stack.

Syntax

```
popd
```

Example

```
user@linux:/var/log$ popd
~/Documents
```

Removes top directory from stack.

QUICK REFERENCE

pwd	Show current directory
ls	List directory contents
cd	Change directory
tree	Show directory tree
clear	Clear terminal screen
history	Show command history
dirs	Show directory stack
pushd	Push directory to stack
popd	Pop directory from stack

PRO TIP

Master these navigation commands to work faster and smarter in Linux!

QUICK TIP

Use Tab key for auto-completion and ↑/↓ arrows to navigate history!

NAVIGATION WORKFLOW

Open Terminal → Explore (ls) → Navigate (cd) → View (tree) → Recall (history) → Done!

Practice daily and become a Linux navigation expert!

Linux™ LIST FILES WITH ls COMMAND

View & Manage Files Like a Pro

Learn | Practice | Use | Explore | Master

WHAT IS ls?

The ls command is used to list files and directories in a directory.

Syntax

```
ls [options] [directory]
```

COMMON FILE TYPES

Regular File	f	file.txt
Directory	d	Documents/
Symbolic Link	l	link -> /usr/bin
Device File	b/c	/dev/sda1
Hidden File	.	.bashrc
Executable File	*	script.sh
Socket File	s	socket.sock
Named Pipe	p	pipe

QUICK TIP

Use Tab key after typing `ls -l` and press Enter to auto-complete!

1. ls

List files and directories in current directory.

```
$ ls
```

Example

```
$ ls
Desktop   Documents   Downloads
Music     Pictures    Videos
file.txt  script.sh   notes.md
```

Shows names only.

2. ls -l

List files in long format (with details).

```
$ ls -l
```

Example

```
$ ls -l
total 32
-rw-r--r--  1 user user   1234 May 18 10:30 file.txt
-rwxr-xr-x  1 user user   2345 May 18 10:31 script.sh
drwxr-xr-x  2 user user   4096 May 18 10:32 Documents
drwxr-xr-x  2 user user   4096 May 18 10:33 Pictures
```

Shows permissions, owner, size, date & time.

3. ls -a

List all files including hidden files.

```
$ ls -a
```

Example

```
$ ls -a
.   ..    .bashrc    .config   .git
Desktop   Documents   file.txt   script.sh
```

Includes hidden files (starting with .)

4. ls -la

List all files (including hidden) in long format.

```
$ ls -la
```

Example

```
$ ls -la
total 56
drwxr-xr-x  5 user user  4096 May 18 10:30 .
drwxr-xr-x 18 root root  4096 May 17 09:10 ..
-rw-r--r--  1 user user   220 May 10 08:20 .bashrc
drwxr-xr-x  3 user user  4096 May 12 11:15 .config
drwxr-xr-x  2 user user  4096 May 18 10:32 Documents
-rw-r--r--  1 user user  1234 May 18 10:30 file.txt
-rwxr-xr-x  1 user user  2345 May 18 10:31 script.sh
```

Combines -l and -a options.

HIDDEN FILES

Files starting with . are hidden by default.

Examples:

```
.bashrc      .profile
.config      .git
.cache       .ssh
.local       .python_history
```

5. ls -lh

List files in long format with human readable sizes.

```
$ ls -lh
```

Example

```
$ ls -lh
total 1.2M
-rw-r--r--  1 user user  1.2K May 18 10:30 file.txt
-rwxr-xr-x  1 user user  2.3K May 18 10:31 script.sh
drwxr-xr-x  2 user user  4.0K May 18 10:32 Documents
drwxr-xr-x  2 user user  4.0K May 18 10:33 Pictures
-rw-r--r--  1 user user  1.1M May 18 10:35 video.mp4
```

Easy to read sizes (K, M, G).

6. ls -R

List files recursively in all subdirectories.

```
$ ls -R
```

Example

```
$ ls -R
.:
Desktop   Documents   file.txt

./Documents:
notes.md   resume.pdf

./Pictures:
photo1.jpg   photo2.jpg
```

Shows files in all subdirectories.

EXTRA OPTIONS

`ls -t`	Sort by modification time (newest first)
`ls -S`	Sort by file size (largest first)
`ls -r`	Reverse order while listing
`ls -h`	Human readable sizes (with -l)
`ls --color=auto`	Colorize the output
`ls -d [dir]`	List directory itself, not contents

Example

```
$ ls -lh --color=auto
(total 1.2M)
drwxr-xr-x 2 user user  4.0K May 18 10:32 Documents
drwxr-xr-x 2 user user  4.0K May 18 10:33 Pictures
-rw-r--r-- 1 user user  1.2K May 18 10:30 file.txt
-rwxr-xr-x 1 user user  2.3K May 18 10:31 script.sh
```

FILE DETAILS (in ls -l)

```
-rw-r--r-- 1 user group 1234 May 18 10:30 file.txt
```

Permissions | Links | Owner | Group | Size | Date | Time | Filename

NOTE

Permissions: r (read), w (write), x (execute)
User Types: - (file), d (directory), l (link)

PRO TIP

Combine options for powerful results!

```
$ ls -lahR --color=auto
```

ls COMMAND WORKFLOW

Open Terminal → Go to Directory → Run ls Command → View Output → Manage Files

Practice these commands daily and become a Linux expert! 💪

Linux™ CREATE FILES & DIRECTORIES

Create Files & Folders Easily in Linux

Learn | Practice | Use | Explore | Master

BASIC CONCEPTS

FILE

A file is a collection of data stored on disk with a specific name.

Examples:
`file.txt, notes.md, script.sh`

DIRECTORY / FOLDER

A directory (folder) is a container that can hold files and other directories.

Examples:
`Documents/, Projects/, Images/`

TIP
In Linux, everything is a file. Even directories are treated as files!

1. touch

touch

Create an empty file or update the timestamp of a file.

Syntax

```
touch [options] filename
```

Example

```
user@linux:~$ touch file.txt
user@linux:~$ touch notes.md
user@linux:~$ touch script.sh
```

Creates an empty file.

2. mkdir

mkdir

Create a new directory (folder).

Syntax

```
mkdir [options] dirname
```

Example

```
user@linux:~$ mkdir MyFolder
user@linux:~$ mkdir Projects
```

Creates a single directory.

3. mkdir -p

mkdir -p

Create parent directories as needed (no error if they already exist).

Syntax

```
mkdir -p path/to/dirname
```

Example

```
user@linux:~$ mkdir -p Projects/2025/May
user@linux:~$ mkdir -p Backup/docs/reports
```

Creates nested directories (parents also).

4. rmdir

rmdir

Remove an empty directory.

Syntax

```
rmdir [options] dirname
```

Example

```
user@linux:~$ rmdir OldFolder
user@linux:~$ rmdir temp
```

Directory must be empty to remove.

COMMON TERMS

File
A file is a unit of data stored on disk.

Directory / Folder
A container that holds files and other folders.

Create
Make a new file or directory.

Path
The location of a file or directory.

EXAMPLES SUMMARY

Command	Purpose	Example	Result
touch	Create a new file	touch file.txt	Creates file.txt
mkdir	Create a new folder	mkdir MyFolder	Creates MyFolder/
mkdir -p	Create nested folders	mkdir -p A/B/C	Creates A/B/C/
rmdir	Remove empty folder	rmdir MyFolder	Removes MyFolder/

PRACTICAL EXAMPLES

1. Create a file
   ```
   user@linux:~$ touch data.txt
   ```
2. Create a folder
   ```
   user@linux:~$ mkdir Docs
   ```
3. Create nested folders
   ```
   user@linux:~$ mkdir -p Docs/Work/Reports
   ```
4. Remove an empty folder
   ```
   user@linux:~$ rmdir OldDocs
   ```

Structure After Step 3

```
Docs/
└── Work/
    └── Reports/
```

All folders are created automatically!

FILE & FOLDER STRUCTURE

```
user@linux:~$ tree -a
.
├── data.txt
├── Docs/
│   └── Work/
│       └── Reports/
└── notes.md

3 directories, 2 files
```

KEY NOTES

- Use `touch` to create files.
- Use `mkdir` to create folders.
- Use `mkdir -p` for nested folders.
- Use `rmdir` only on empty folders.
- Linux is case-sensitive. (File.txt ≠ file.txt)

PRO TIP
Use Tab key for auto-completion and faster typing! [Tab]

CREATE WORKFLOW

Open Terminal → Create File (touch) → Create Folder (mkdir) → Create Nested Folder (mkdir -p) → Remove Empty Folder (rmdir) → Done!

Practice daily and master Linux file & directory management!

Linux™ COPY & MOVE COMMANDS

Copy, Move & Rename Files and Directories Like a Pro

Learn | Practice | Use | Explore | Master

WHAT YOU CAN DO

COPY (cp)
Make a copy of files or directories.

COPY RECURSIVE (cp -r)
Copy directories and their contents recursively.

MOVE (mv)
Move or rename files and directories.

RENAME (rename)
Rename multiple files using a pattern.

1. cp (COPY FILES)

cp

Copy files from source to destination.

Syntax

```
cp [options] source destination
```

Example

```
user@linux:~$ cp file.txt backup.txt
user@linux:~$ cp notes.md /home/user/docs/
```

Use Case

Copy a file to another location or with a new name.

2. cp -r (COPY DIRECTORIES)

cp -r

Copy directories recursively (with all files and subdirectories).

Syntax

```
cp -r [options] source_dir destination_dir
```

Example

```
user@linux:~$ cp -r MyFolder BackupFolder
user@linux:~$ cp -r Project /home/user/docs/
```

Use Case

Make a full backup of a directory including all contents.

3. mv (MOVE / RENAME)

mv

Move files or directories to a new location or rename them.

Syntax

```
mv [options] source destination
```

Examples

```
user@linux:~$ mv file.txt /home/user/docs/
user@linux:~$ mv oldname.txt newname.txt
```

Use Case

Move a file/folder to another location or rename a file/folder.

4. rename (RENAME MULTIPLE FILES)

A→B rename

Rename multiple files using pattern (Perl compatible).

Syntax

```
rename 's/old/new/' files...
```

Examples

```
user@linux:~$ rename 's/old/new/' *.txt
user@linux:~$ rename 's/.png/_image.png/' *
```

Use Case

Rename many files at once using search and replace pattern.

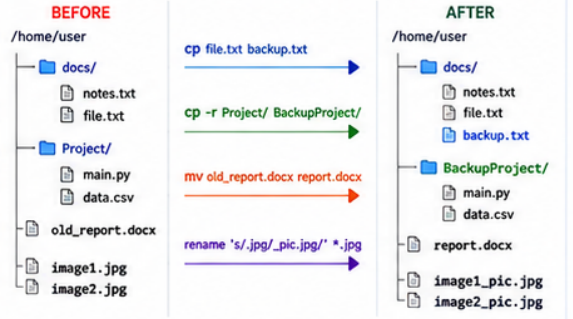
EMPORPLES IN ACTION

Command	What it does	Example
`cp file.txt backup.txt`	Copy file with new name	`user@linux:~$ cp file.txt backup.txt`
`cp file.txt /home/user/docs/`	Copy file to another folder	`user@linux:~$ cp file.txt /home/user/docs/`
`cp -r MyFolder Backup/`	Copy directory recursively	`user@linux:~$ cp -r MyFolder Backup/`
`mv file.txt /home/user/docs/`	Move file to another folder	`user@linux:~$ mv file.txt /home/user/docs/`
`mv old.txt new.txt`	Rename file	`user@linux:~$ mv old.txt new.txt`
`mv MyFolder NewFolder`	Rename / Move directory	`user@linux:~$ mv MyFolder NewFolder`
`rename 's/img/photo/' *.jpg`	Rename many files	`user@linux:~$ rename 's/img/photo/' *.jpg`

SAMPLE SCENARIO

1. Copy notes.txt to backup.txt
   ```
   user@linux:~$ cp notes.txt backup.txt
   ```
2. Copy project folder to backup folder
   ```
   user@linux:~$ cp -r Project Backup/
   ```
3. Move report.docx to Documents folder
   ```
   user@linux:~$ mv report.docx /home/user/Documents/
   ```
4. Rename old_report.docx to new_report.docx
   ```
   user@linux:~$ mv old_report.docx new_report.docx
   ```
5. Rename all .jpg to _pic.jpg
   ```
   user@linux:~$ rename 's/.jpg/_pic.jpg/' *.jpg
   ```

BEFORE & AFTER



QUICK REFERENCE

- `cp` Copy files
- `cp -r` Copy directories
- `mv` Move / Rename
- `rename` Rename multiple files

TIP
Practice these commands regularly to handle files like a Linux expert!

PRO TIP

Always double-check the source and destination before running commands.

COPY & MOVE WORKFLOW

Identify Source → Choose Command (cp / mv / rename) → Set Destination → Execute → Verify Result

Master file management today and boost your Linux skills! 💪

Linux™ DELETE FILES COMMANDS

Delete / Remove Files & Directories Safely in Linux

Learn | Practice | Use | Explore | Master

WHAT DOES DELETE MEAN?

Delete (or Remove) means permanently removing files or directories from the filesystem.

- File → A single data item.
- Directory → A container for files and other folders.
- Delete → Remove permanently from system.

TIP
Deleted files go to Trash in GUI, but in Terminal, deleted files are permanently removed.

1. rm (REMOVE FILES)

`rm` — Remove files.

Syntax

```
rm [options] file_name
```

Example

```
user@linux:~$ rm file.txt
user@linux:~$ rm notes.md
```

Removes a file permanently.

2. rm -r (REMOVE DIRECTORIES)

`rm -r` — Remove directories recursively.

Syntax

```
rm -r [directory_name]
```

Example

```
user@linux:~$ rm -r MyFolder
user@linux:~$ rm -r Projects/
```

Removes a directory and all its contents.

3. rm -f (FORCE DELETE)

`rm -f` — Force remove files without asking for confirmation.

Syntax

```
rm -f file_name
```

Example

```
user@linux:~$ rm -f file.txt
user@linux:~$ rm -f notes.md
```

Force deletes the file without any prompt.

4. rm -rf (FORCE REMOVE DIR)

`rm -rf` — Force remove directory recursively without prompt.

Syntax

```
rm -rf directory_name
```

Example

```
user@linux:~$ rm -rf MyFolder
user@linux:~$ rm -rf Project/
```

Be very careful! Deletes without confirmation.

5. rmdir (REMOVE EMPTY DIR)

`rmdir` — Remove empty directory only.

Syntax

```
rmdir directory_name
```

Example

```
user@linux:~$ rmdir EmptyFolder
user@linux:~$ rmdir Temp/
```

Directory must be empty, otherwise it will show error.

COMMAND QUICK REFERENCE

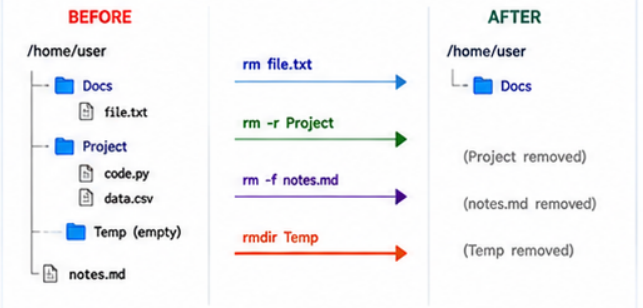
Command	What it does	Use for	Danger Level
`rm`	Remove files	Delete single file	Low
`rm -r`	Remove directories recursively	Delete folder & all contents	Medium
`rm -f`	Force remove files	Delete file without prompt	High
`rm -rf`	Force remove directories	Delete folder & all contents without prompt	Very High
`rmdir`	Remove empty directory	Delete empty folder only	Low

WARNING
Once a file is deleted using `rm`, it cannot be recovered easily. Always double-check before running `rm -rf` !

PRACTICAL EXAMPLES

1. Delete file.txt
```
user@linux:~$ rm file.txt
```
2. Delete notes.md (force)
```
user@linux:~$ rm -f notes.md
```
3. Delete MyFolder with all files
```
user@linux:~$ rm -r MyFolder
```
4. Force delete Project folder (no prompt)
```
user@linux:~$ rm -rf Project
```
5. Remove empty folder Temp
```
user@linux:~$ rmdir Temp
```

BEFORE & AFTER (Example)



BEST PRACTICES

- Always check file/directory name before deleting.
- Use `rm -i` for interactive delete (one by one confirm).
- Backup important data before running `rm -rf`.
- Use `ls` to verify before delete.
- Think before you delete! 😊

PRO TIP
Use Tab key for auto-completion and avoid typing mistakes.

PRO TIP
Always preview with `ls` before deleting anything!

`ls → check → rm`

DELETE WORKFLOW

1. List Files / Folders (`ls`) → 2. Verify Name (double-check) → 3. Choose Right Command (`rm`) → 4. Be Careful 😎 (Especially `rm -rf`) → 5. Done!

Practice daily and become a Linux command master! 💪🐧

Linux™ VIEW FILE CONTENT COMMANDS

View & Read File Content in Linux Like a Pro

Learn | Practice | Use | Explore | Master

WHAT IS FILE CONTENT?

Linux provides many commands to view and read file content in different ways.

USEFUL FOR

- Reading configuration files
- Checking logs
- Viewing code/scripts
- Debugging & monitoring

TIP

Use less for large files, cat for small files, and tail -f for live logs.

1. cat (CONCATENATE)

cat

Display entire file content from beginning to end.

Syntax

```
cat [file_name]
```

Example

```
user@linux:~$ cat file.txt
Hello Linux!
This is a sample file.
Have a great day!
```

Best for small files.

2. tac (REVERSE CAT)

tac

Display entire file content in reverse order (line by line).

Syntax

```
tac [file_name]
```

Example

```
user@linux:~$ tac file.txt
a equal taerg a evaH
.elif elpmas a si sihT
!xuniL olleH
```

Reverse of cat.

3. less (VIEW PAGER)

less

View file content page by page. Allows scrolling up and down.

Syntax

```
less [file_name]
```

Example

```
user@linux:~$ less largefile.log
(Use arrows / PageUp / PageDown)
Press q to quit
```

Best for large files.

4. more (SIMPLE PAGER)

more

View file content page by page (forward only).

Syntax

```
more [file_name]
```

Example

```
user@linux:~$ more file.txt
-- More -- (press Enter)
(Press q to quit)
```

Older than less, limited features.

5. head (START OF FILE)

head

Display first few lines of a file (default 10).

Syntax

```
head [file_name]
# head -n [number] [file]
```

Example

```
user@linux:~$ head file.txt
Hello Linux!
This is a sample file.
Have a great day!
```

Use -n to specify number of lines.

6. tail (END OF FILE)

tail

Display last few lines of a file (default 10).

Syntax

```
tail [file_name]
# tail -n [number] [file]
# tail -f [file] (follow)
```

Example

```
user@linux:~$ tail file.txt
Keep learning Linux.
Stay consistent.
You will succeed!
```

Use -f to watch live updates (like logs).

7. nl (NUMBER LINES)

nl

Number all lines of a file.

Syntax

```
nl [file_name]
# nl -ba [file_name]  (with blank lines)
```

Example

```
user@linux:~$ nl file.txt
     1  Hello Linux!
     2  This is a sample file.
     3  Have a great day!
```

```
user@linux:~$ nl -ba file.txt
     1  Hello Linux!
     2  This is a sample file.
     3  Have a great day!
```

Very useful for scripts, logs & documentation.

COMMAND COMPARISON

Command	Purpose	Direction	Pagination	Best For
cat	Show entire file	Normal	No	Small files
tac	Show entire file (reverse)	Reverse	No	Reverse reading
less	View with navigation	Both	Yes	Large files
more	View forward only	Forward	Yes	Simple use
head	Show first lines	Top	No	Start of file
tail	Show last lines	Bottom	No	End of file / Logs
nl	Number lines	Normal	No	Scripts / Logs

PRACTICAL EXAMPLES

1. View small file `cat file.txt`
2. View large file with scroll `less bigfile.log`
3. View last 20 lines /ɯɯr/log/syslog
4. Wat.-n 2f live (lx
4. Viech log live t-f /ɯɯr/lɔɔɯ/lɔg 20
5. View first 5 lines. txt
6. Number all lines nl file.txt
7. View file in reverse tac file.txt

QUICK REFERENCE

cat	Show entire file
tac	Show entire file in reverse
less	Scrollable viewer (up/down)
more	Scroll forward only
head	Show first N lines
tail	Show last N lines / follow
nl	Number all lines

NOTE

Press q to quit less or more.
Use arrow keys in less to navigate.

PRO TIP

Remember: less is your best friend for large files!

VIEW FILE CONTENT WORKFLOW

Choose File → Pick Command (cat/less/tail etc.) → Run Command → View Content → Understand & Analyze

Master these commands and read any file like a pro! 💪

Linux™ FILE SEARCH COMMANDS

Find Files, Directories & Commands in Linux Like a Pro

Learn | Practice | Use | Explore | Master

WHAT IS FILE SEARCH?

Linux provides powerful commands to search files, directories and commands quickly.

- **find** Search files & directories by name, size, date, etc.
- **locate** Search files using an updated database.
- **which** Show the path of a command.
- **whereis** Locate binary, source and man page of a command.
- **type** Show how a command will be interpreted.

TIP
Use locate for fast search and find for advanced search with conditions.

1. find (POWERFUL SEARCH)

find
Search files and directories recursively with filters.

Syntax

```
find [path] [options] [expression]
```

Examples

```
# Find by name
find . -name "file.txt"
# Find by type
find /home/user -type d
# Find by size (+100M)
find . -size +100M
```

Most powerful but slower.

2. locate (FAST SEARCH)

locate
Search files using an indexed database. Very fast!

Syntax

```
locate [filename]
```

Examples

```
locate file.txt
locate notes.pdf
locate "*.log"
```

Database updated by 'updatedb' (locate may not find new files).

3. which (COMMAND PATH)

which
Show the full path of a command that will be executed.

Syntax

```
which [command]
```

Examples

```
which python
which gcc
which ls
```

Shows path of command in $PATH.

4. whereis (COMMAND INFO)

whereis
Locate binary, source and man page of a command.

Syntax

```
whereis [command]
```

Examples

```
whereis python
whereis gcc
whereis ls
```

Shows binary, source and man page locations.

5. type (COMMAND TYPE)

type
Show how a command will be interpreted by the shell.

Syntax

```
type [command]
```

Examples

```
type ls
type cd
type python
```

Tells if it is alias, builtin, function or external command.

FIND COMMON OPTIONS

Option	Meaning	Example
-name	Search by filename	-name "file.txt"
-type	f = file, d = directory	-type f
-size	Search by size	-size +100M
-mtime	Modified in N days	-mtime -7
-user	Search by user	-user username
-perm	Search by permission	-perm 644

Run 'man find' for more options.

UPDATE LOCATE DATABASE

locate uses a database. Update it regularly.

```
sudo updatedb
```

Or add to cron for auto update:

```
sudo crontab -e
0 3 * * * /usr/bin/updatedb
```

Updatedb may take time on first run.

SEARCH EXAMPLES (QUICK REFERENCE)

Task	Command
Find file by name	find . -name "file.txt"
Find directory by name	find . -type d -name "docs"
Find files larger than 100MB	find . -type f -size +100M
Find files modified in last 7 days	find . -mtime -7
Fast search file using locate	locate file.txt
Find command path	which python
Find command info	whereis gcc
Check command type	type ls

SAMPLE SCENARIO

1. You want to find "report.pdf" in home directory.
   `find /home/user -name "report.pdf"`
2. You want faster search (if DB updated).
   `locate report.pdf`
3. You want to know which command "git" will run.
   `which git`
4. You want details of "git" command.
   `whereis git`
5. You want to see how "ls" works.
   `type ls`

PRO TIP
Use locate for speed, find for power! 🚀

FILE SEARCH WORKFLOW

Know What to Search → Use locate (Fast) → Use find (Advanced) → Find Command Path (which) → More Info (whereis) → Check Type (type) → Done!

Master file search commands and save time in Linux! 💪🐧

Linux™ TEXT SEARCH COMMANDS

Search Text Like a Pro with grep

Learn | Practice | Use | Explore | Master

WHAT IS TEXT SEARCH?

grep is a powerful command-line tool used to search for patterns in text within files.

COMMON USES

- Search logs and reports
- Find text in files
- Filter command output
- Code & config search
- Troubleshooting

TIP
grep stands for Global Regular Expression Print.

1. grep (BASIC SEARCH)

grep

Search for a pattern in a file and print matching lines.

Syntax

```
grep [pattern] [file]
```

Example

```
user@linux:~$ grep "error" log.txt
Error: failed to load module
Error: something went wrong
Error: connection failed
```

Search is case-sensitive.

2. grep -i (IGNORE CASE)

grep -i

Search for a pattern ignoring case (upper/lower case).

Syntax

```
grep -i [pattern] [file]
```

Example

```
user@linux:~$ grep -i "error" log.txt
Error: failed to load module
error: something went wrong
ERROR: connection failed
```

Matches 'Error', 'error', 'ERROR', etc.

3. grep -r (RECURSIVE SEARCH)

grep -r

Search for a pattern recursively in a directory and its subdirectories.

Syntax

```
grep -r [pattern] [directory]
```

Example

```
user@linux:~$ grep -r "TODO" ./project/
./project/app.py:  # TODO: fix bug
./project/utils/helper.js: // TODO: refactor
./project/readme.md: TODO list
```

Searches all files in the directory recursively.

4. grep -n (SHOW LINE NUMBER)

grep -n

Search for a pattern and show line numbers of matching lines.

Syntax

```
grep -n [pattern] [file]
```

Example

```
user@linux:~$ grep -n "error" log.txt
12:Error: failed to load module
45:Error: something went wrong
78:Error: connection failed
```

Very useful to quickly jump to the exact line.

5. grep -v (INVERT MATCH)

grep -v

Print lines that do NOT match the pattern.

Syntax

```
grep -v [pattern] [file]
```

Example

```
user@linux:~$ grep -v "debug" log.txt
Info: system started
Warning: high memory usage
Error: connection failed
```

Shows lines that do not contain the given pattern.

OTHER USEFUL OPTIONS

Option	Meaning	Example
-c	Count matching lines	grep -c "error" log.txt
-l	Show file name only	grep -l "main" *.py
-w	Match whole word	grep -w "cat" file.txt
-E	Use extended regex	grep -E "err(or|or)" file.txt
--color=auto	Highlight matches	grep --color=auto "err" file.txt

Use 'man grep' to explore all options.

SEARCH PATTERNS (EXAMPLES)

Pattern	Matches
error	Lines containing 'error'
^error	Lines starting with 'error'
error$	Lines ending with 'error'
err.r	Matches 'error', 'err0r', etc.
err(or|or)	Matches 'error' or 'err0r'
[0-9]+	Lines containing one or more digits

PRACTICAL EXAMPLES

1. Search for 'error' in file
   ```
   user@linux:~$ grep "error" log.txt
   ```
2. Search 'error' ignoring case
   ```
   user@linux:~$ grep -i "error" log.txt
   ```
3. Search 'TODO' in all files under ./src
   ```
   user@linux:~$ grep -r "TODO" ./src/
   ```
4. Search 'fail' and show line numbers
   ```
   user@linux:~$ grep -n "fail" log.txt
   ```
5. Show lines that do NOT contain 'debug'
   ```
   user@linux:~$ grep -v "debug" log.txt
   ```

SAMPLE SCENARIO

You have a log file and want to find errors, warnings, and exclude debug lines.

```
# Find errors (case-insensitive)
grep -i "error" server.log
# Show line numbers for warnings
grep -n "warning" server.log
# Show lines excluding debug info
grep -v "debug" server.log
```

PRO TIP
Combine grep with pipes (|) to search command output.
Example: `ps aux | grep nginx`

PRO TIP
Master grep and you can find anything, anywhere in seconds!

TEXT SEARCH WORKFLOW

Choose File or Directory → Define Pattern (What to Search) → Run grep Command with Options → Review Results → Refine Search (if needed) → Done!

Practice daily and become a Linux search ninja!

Linux™ TEXT PROCESSING COMMANDS

Transform, Filter & Process Text Like a Pro

Learn | Practice | Use | Explore | Master

WHAT IS TEXT PROCESSING?

Text processing commands help you manipulate, filter and transform text data from files or input.

COMMON USES

- Edit and transform text
- Extract columns and fields
- Sort and remove duplicates
- Replace and translate text
- Process logs and reports

TIP
These commands can be combined using pipes (|) for powerful text processing.

1. sed (STREAM EDITOR)

sed — Stream editor for filtering and transforming text.

Syntax

```
sed 'options' pattern file
```

Example

```
# Replace 'old' with 'new'
sed 's/old/new/g' file.txt
# Delete line 3
sed '3d' file.txt
```

Powerful for find & replace, delete, insert, and more.

2. awk (PATTERN SCANNING)

awk — Pattern scanning and processing language.

Syntax

```
awk 'pattern {action}' file
```

Example

```
# Print columns
awk '{print $1, $3}' file.txt
# Sum of numbers in column 2
awk '{sum+=$2} END {print sum}' file.txt
```

Great for complex patterns and calculations.

3. cut (SELECT COLUMNS)

cut — Remove sections from each line of files.

Syntax

```
cut -d 'delim' -f fields file
```

Example

```
# Cut by column (,) and print 1st,3rd
cut -d ',' -f 1,3 file.csv
# Cut characters 1-5
cut -c 1-5 file.txt
```

Useful for extracting columns or specific parts.

4. sort (SORT LINES)

sort — Sort lines of text files alphabetically or numerically.

Syntax

```
sort [options] file
```

Example

```
# Sort ascending
sort file.txt
# Sort numeric
sort -n numbers.txt
# Sort reverse
sort -r file.txt
```

Sort data, numbers, dates and more.

5. uniq (UNIQUE LINES)

uniq — Report or filter out repeated lines.

Syntax

```
uniq [options] file
```

Example

```
# Remove duplicates (adjacent)
sort file.txt | uniq
# Show count of each line
sort file.txt | uniq -c
```

Works best with sorted input.

6. tr (TRANSLATE CHARACTERS)

tr — Translate or delete characters.

Syntax

```
tr 'set1' 'set2'
```

Example

```
# Lowercase to uppercase
tr 'a-z' 'A-Z' < file.txt
# Delete digits
tr -d '0-9' < file.txt
```

Translate, delete or squeeze characters.

SAMPLE DATA (employees.txt)

ID	Name	Dept	Salary	JoinDate
101	Arjun	IT	55000	2024-01-10
102	Neha	HR	45000	2023-11-05
103	Rohan	IT	62000	2023-09-12
104	Priya	Finance	48000	2024-02-20
105	Arjun	IT	55000	2024-01-10
106	Karan	HR	45000	2023-11-05

File: employees.txt (CSV format)
Delimiter (,)

PRACTICAL EXAMPLES

1. Replace 'IT' with 'Technology'
   `sed 's/IT/Technology/g' employees.txt`
2. Print Name and Salary columns
   `awk -F, '{print $2,$4}' employees.txt`
3. Get Name, Dept and Salary columns
   `cut -d',' -f2,3,4 employees.txt`
4. Sort by Name (A-Z)
   `sort -t',' -k2 employees.txt`
5. Remove duplicate rows
   `sort employees.txt | uniq`
6. Convert lowercase to UPPERCASE
   `tr 'a-z' 'A-Z' < employees.txt`

COMMAND COMPARISON

Command	Purpose	Works On	Key Option Example
sed	Edit & replace text	Lines	s/old/new/, d, i
awk	Pattern scan & process	Fields	-F',', {print}, END
cut	Extract columns/parts	Fields/Chars	-d',' -f1,3 or -c1-5
sort	Sort lines	Lines	-n (numeric), -r (reverse)
uniq	Remove duplicates	Lines	-c (count), -u (unique)
tr	Translate/delete chars	Characters	-d, a-z A-Z

Combine these commands using pipes (|) for powerful one-liners!

QUICK REFERENCE

sed	stream editor
awk	pattern scanning & processing
cut	cut (extract) sections
sort	sort lines
uniq	unique lines
tr	translate characters

TIP
Use man <command> to learn more about options and advanced usage.

SAMPLE OUTPUT (After Processing)

```
# After sed replace IT with Technology
101,Arjun,Technology,55000,2024-01-10
...
# After awk print Name and Salary
Arjun 55000
Neha  45000
...
# After sort by Name
101,Arjun,IT,55000,2024-01-10
105,Arjun,IT,55000,2024-01-10
...
```

PRO TIP
Master pipes (|) to combine commands and save time!

```
cat file.txt | grep error | sort | uniq -c
```

TEXT PROCESSING WORKFLOW

Input File (Text Data) → Search / Filter (grep / sed) → Extract / Cut (cut / awk) → Sort Data (sort) → Remove Duplicates (uniq) → Transform (tr / sed) → Done! (Clean Output)

Practice daily, automate tasks and become a Linux pro!

Linux™ FILE PERMISSIONS COMMANDS

Manage File Permissions, Ownership & Access in Linux

Learn | Practice | Use | Explore | Master

WHAT IS FILE PERMISSIONS?

File permissions control who can read, write or execute a file or directory.

THREE TYPES OF ACCESS

- **r** Read — View file content or list directory.
- **w** Write — Modify file content or create/delete inside directory.
- **x** Execute — Run file as program or enter directory.

TIP
Permissions are shown in the format: rwx (read, write, execute).

1. chmod (CHANGE MODE)

chmod — Change file or directory permissions (mode).

Syntax

```
chmod [options] mode file
```

Examples

```
chmod 755 file.sh      # rwxr-xr-x
chmod 644 file.txt     # rw-r--r--
chmod -R 755 folder/   # recursive
```

Numbers: r=4, w=2, x=1
Example: 7 = 4+2+1 (rwx)
5 = 4+1 (r-x)
6 = 4+2 (rw-)

2. chown (CHANGE OWNER)

chown — Change owner (user) of a file or directory.

Syntax

```
chown [options] user file
```

Examples

```
chown user file.txt
chown -R user:group folder/
chown user:group file.sh
```

Change who owns the file. Use root or sudo for other users.

3. chgrp (CHANGE GROUP)

chgrp — Change group ownership of a file or directory.

Syntax

```
chgrp [options] group file
```

Examples

```
chgrp group file.txt
chgrp -R group folder/
```

Change the group that owns the file. Use root or sudo.

PERMISSIONS OVERVIEW (r, w, x)

Permission	Symbol	Value	Meaning
r	Read	4	View file content or list directory
w	Write	2	Modify file or create / delete files
x	Execute	1	Execute file or enter directory
-	None	0	No permission

PERMISSION VALUES

r (4)	w (2)	x (1)	
rwx	= 7	(4+2+1)	Read, Write, Execute
rw-	= 6	(4+2+0)	Read, Write
r-x	= 5	(4+0+1)	Read, Execute
r--	= 4	(4+0+0)	Read only
-w-	= 2	(0+2+0)	Write only
--x	= 1	(0+0+1)	Execute only
---	= 0	(0+0+0)	No permission

PERMISSION STRUCTURE

ls -l output example:

```
-rwxr-xr--   1 user group   1024  May 20 10:00 file.txt
```

File Type	User (user)	Group (group)	Others (others)
- regular file d directory l symbolic link	rwx (7) Owner permissions	r-x (5) Group permissions	r-- (4) Others permissions

WHO HAS ACCESS?

- **User (Owner)** — The file owner (U). First 3 permissions.
- **Group** — Users in the file's group (G). Next 3 permissions.
- **Others** — Everyone else on the system (O). Last 3 permissions.

NUMERIC EXAMPLES

```
chmod 755 file.sh      → rwxr-xr-x  (7 5 5)
chmod 644 file.txt     → rw-r--r--  (6 4 4)
chmod 700 secret.sh    → rwx------  (7 0 0)
chmod -R 775 folder/   → rwxrwxr-x  (7 7 5)
```

PRACTICAL EXAMPLES

1. Make script executable by owner
   `chmod 700 script.sh`
2. Allow all to read, owner to write
   `chmod 644 file.txt`
3. Allow owner & group full access, others read
   `chmod 764 file.txt`
4. Allow all full access (not recommended)
   `chmod 777 file.txt`
5. Set directory with full access for user & group
   `chmod 775 folder/`

COMMON PERMISSIONS QUICK GUIDE

Numeric	Symbolic	Meaning	Use Case
700	rwx------	Owner full access	Private scripts
644	rw-r--r--	Owner write, all read	Text files
755	rwxr-xr-x	Owner full, others read & execute	Programs / Scripts
775	rwxrwxr-x	Owner & group full, others read & execute	Shared folders
777	rwxrwxrwx	All users full access	Public access (Not secure)

WARNING
Avoid using 777 in production. It can be a security risk!

PRO TIP
Use 755 for scripts and executables.
Use 644 for files and documents.

```
chmod 755 script.sh
```

FILE PERMISSIONS WORKFLOW

1. Check Permissions — ls -l file
2. Change Permissions — chmod mode file
3. Change Owner — chown user file
4. Change Group — chgrp group file
5. Verify Changes — ls -l file

Understand permissions, secure your files, stay safe! 💪🐧

Linux™ PERMISSION NUMBERS

Understand chmod Numbers & File Permissions

Learn | Practice | Use | Explore | Master

WHAT ARE PERMISSION NUMBERS?

Linux permissions can also be represented using numbers. Each permission has a value, and the total value gives the permission number.

BASIC PERMISSIONS & VALUES

Permission	Symbol	Value	Meaning
r	Read	4	View file content or list directory
w	Write	2	Modify file or create/delete
x	Execute	1	Run file or enter directory
-	None	0	No permission

TIP
Add the values to get the permission number (0-7) for each category.

1. chmod 755 (RWX R-X R-X)

```
chmod 755 file.sh
```

Breakdown

User (Owner)	rwx (7)
Group	r-x (5)
Others	r-x (5)

Numeric: 7 5 5

Meaning

- Owner: Read, Write, Execute
- Group: Read, Execute
- Others: Read, Execute
- Common for scripts & programs

2. chmod 644 (RW- R-- R--)

```
chmod 644 file.txt
```

Breakdown

User (Owner)	rw- (6)
Group	r-- (4)
Others	r-- (4)

Numeric: 6 4 4

Meaning

- Owner: Read, Write
- Group: Read only
- Others: Read only
- Common for text & documents

3. chmod 700 (RWX --- ---)

```
chmod 700 private.sh
```

Breakdown

User (Owner)	rwx (7)
Group	--- (0)
Others	--- (0)

Numeric: 7 0 0

Meaning

- Owner: Read, Write, Execute
- Group: No permission
- Others: No permission
- Private file (only owner access)

HOW TO CALCULATE PERMISSION NUMBERS

Add the values of r, w, x for each category.

User (Owner)	Group	Others
r = 4	r = 4	r = 4
w = 2	w = 2	w = 2
x = 1	x = 1	x = 1
Total = 7	Total = 5	Total = 5

Example: 755

QUICK VALUE REFERENCE

Permission	Symbolic	Numeric
No permission	---	0
Execute only	--x	1
Write only	-w-	2
Write + Execute	-wx	3
Read only	r--	4
Read + Execute	r-x	5
Read + Write	rw-	6
Read + Write + Execute	rwx	7

SEE PERMISSIONS WITH ls -l

```
$ ls -l
total 4
-rwxr-xr-x  1 user group  1234 May 20 10:00 file.sh
-rw-r--r--  1 user group   567 May 20 10:05 file.txt
rwx------   1 user group   890 May 20 10:10 private.sh
```

File Type	User (Owner)	Group	Others
- = file	rwx = 7	r-x = 5	r-x = 5
d = directory	rw- = 6	r-- = 4	r-- = 4
	rwx = 7	--- = 0	--- = 0

NUMBER TO PERMISSION TABLE

Number	Symbolic	Meaning
0	---	No permission
1	--x	Execute only
2	-w-	Write only
3	-wx	Write + Execute
4	r--	Read only
5	r-x	Read + Execute
6	rw-	Read + Write
7	rwx	Read + Write + Execute

COMMON PERMISSION SETS

Permission	Symbolic	Use Case
755	rwxr-xr-x	Scripts, Programs
644	rw-r--r--	Text files, Docs
700	rwx------	Private files
600	rw-------	Private data
777	rwxrwxrwx	Full access (Not secure)
666	rw-rw-rw-	Read & Write for all
555	r-xr-xr-x	Read & Execute for all

IMPORTANT NOTES

- Avoid using 777 (full access) for security.
- Use minimum required permissions.

NOTE

- You can also use symbolic method:
  `chmod u=rwx,g=rx,o=rx file.sh`
  (`chmod 755 file.sh`)

PRO TIP
Memorize permission values:
4 (Read) + 2 (Write) + 1 (Execute)
Makes Linux permissions super easy!

PERMISSION NUMBER WORKFLOW

Check Current Permissions (`ls -l`) → Calculate Values (r=4, w=2, x=1) → Set Permission (Example: 7 5 5) → Apply Command (`chmod 755 file.sh`) → Verify Changes (`ls -l`)

Secure permissions, secure system!

Linux™ USERS & GROUPS COMMANDS

Manage Users, Groups & Permissions Like a Pro

Learn | Practice | Use | Explore | Master

1. whoami (WHO AM I)

Show current logged-in user.

Syntax

```
whoami
```

Example

```
user@linux:~$ whoami
user
```

Shows your current username.

2. who (WHO IS LOGGED IN)

Show who is currently logged in.

Syntax

```
who
```

Example

```
user@linux:~$ who
user     pts/0        2024-05-20 10:10
jeet     pts/1        2024-05-20 10:15
```

Displays all logged in users.

3. id (USER ID INFO)

Show user and group ID information.

Syntax

```
id [username]
```

Example

```
user@linux:~$ id
uid=1000(user) gid=1000(user)
groups=1000(user),24(cdrom),27(sudo)
```

Shows UID, GID and group info.

4. users (LOGGED IN USERS)

List all users currently logged in.

Syntax

```
users
```

Example

```
user@linux:~$ users
user jeet
```

Quick list of logged in users.

5. groups (SHOW GROUPS)

Show groups of current user.

Syntax

```
groups [username]
```

Example

```
user@linux:~$ groups
user : user cdrom sudo dip plugdev
```

Shows your group memberships.

6. adduser (ADD USER - DEBIAN/UBUNTU)

Add a new user with home directory.

Syntax

```
sudo adduser username
```

Example

```
user@linux:~$ sudo adduser newuser
Adding user 'newuser' ... Done
```

Interactive process, asks for details.

7. useradd (ADD USER - RHEL/CENTOS)

Create a new user (manual method).

Syntax

```
sudo useradd [options] username
```

Example

```
user@linux:~$ sudo useradd -m -s /bin/bash newuser
```

Use `-m` for home, `-s` for shell.

8. passwd (SET/CHANGE PASSWORD)

Set or change user password.

Syntax

```
passwd [username]
```

Example

```
user@linux:~$ passwd newuser
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Use without username to change your own pacsword.

COMMON OPTIONS

Command	Option	Meaning
useradd	-m	Create home directory
	-s /bin/bash	Set default shell
	-G group	Add user to supplementary group
	-u UID	Specify user ID
adduser	--disabled-password	Create user without password
	--gecos "Name"	Set full name / description
passwd	-l	Lock user account
	-u	Unlock user accuont
	-d	Delete user password

EXAMPLE: CREATE & MANAGE USER

1 sudo adduser demo	Create user 'demo' (Ubuntu/Debian)
2 sudo useradd -m -s /bin/bash demo2	Create user 'demo2' (RHEL/CentOS)
3 sudo passwd demo	Set password for 'demo'
4 sudo usermod -aG sudo demo	Add 'demo' to sudo group
5 id demo	Check user info
6 groups demo	Check user groups

IMPORTANT FILES

File	Purpose
/etc/passwd	Stores user account information
/etc/shadow	Stores encrypted passwords
/etc/group	Stores group information
/etc/gshadow	Stores group passwords
/etc/sudoers	Sudo (admin) permissions

USER TYPES

- **Regular User** — Normal user with limited access
- **Super User (root)** — Full control over the system
- **System User** — Used by system services
- **Group** — Collection of users

QUICK REFERENCE TABLE

Command	Description
whoami	Show current user
who	Show who is logged in
id	Show user & group IDs
users	List logged in users
groups	Show groups of user
adduser	Add user (Debian/Ubuntu)
useradd	Add user (RHEL/CentOS)
passwd	Set or change password

REMOVE USER (EXTRA)

Delete a user and home directory.

Syntax

```
sudo userdel -r username
```

Example

```
user@linux:~$ sudo userdel -r demo
```

Use `-r` to remove home directory.

PRO TIP

Use strong passwords and create users only when needed! 🔒

USER MANAGEMENT WORKFLOW

Add User (adduser / useradd) → Set Password (passwd) → Add to Groups (usermod -aG) → Check Info (id / groups) → Done!

Manage users & groups like a pro, keep your system secure! 🔒

Linux™ GROUP MANAGEMENT COMMANDS

Manage Groups & Group Memberships Like a Pro

Learn | Practice | Use | Explore | Master

1. groupadd (ADD GROUP)

Create a new group.

Syntax

```
groupadd [options] groupname
```

Example

```
user@linux:~$ sudo groupadd devteam
Group 'devteam' created
```

Add a new group to the system.

2. groupdel (DELETE GROUP)

Delete an existing group.

Syntax

```
groupdel groupname
```

Example

```
user@linux:~$ sudo groupdel devteam
Group 'devteam' removed
```

Delete a group from the system.

3. usermod (MODIFY USER)

Modify a user account, including group membership.

Syntax

```
usermod [options] username
```

Example

```
user@linux:~$ sudo usermod -aG devteam jeet
Add user 'jeet' to group 'devteam'
```

Use -aG to append user to a group.

4. gpasswd (GROUP ADMIN)

Administer group membership and password.

Syntax

```
gpasswd [options] groupname
```

Example

```
user@linux:~$ sudo gpasswd -a jeet devteam
Adding user jeet to group devteam
```

Add (-a), remove (-d) or set password.

5. groups (SHOW GROUPS)

Show groups for the current user or another user.

Syntax

```
groups [username]
```

Example

```
user@linux:~$ groups jeet
jeet : jeet sudo devteam students
```

List all groups for a user.

6. /etc/group (GROUP FILE)

System file that stores all group information.

Syntax

```
cat /etc/group
```

Example

```
user@linux:~$ cat /etc/group
root:x:0:
sudo:x:27:
devteam:x:1500:jeet,aman
```

Contains all groups on the system.

/etc/group FILE FORMAT

Each line in `/etc/group` file has 4 fields separated by ':'

```
groupname:x:GID:user1,user2,user3
```

Field	Description	Example
groupname	Name of the group	devteam
x	Password placeholder (x)	x
GID	Group ID (number)	1500
user1,user2	Comma-separated users	jeet,aman

TIP
System groups usually have low GID (0-999).
Custom groups start from GID 1000+.

COMMON GROUP MANAGEMENT EXAMPLES

1. Create a group	sudo groupadd designers
2. Delete a group	sudo groupdel designers
3. Add user to group	sudo usermod -aG designers jeet
4. Remove user from group	sudo gpasswd -d jeet designers
5. Set password for group	sudo gpasswd designers
6. Show current user groups	groups
7. Show groups of user 'jeet'	groups jeet

GROUP TYPES

Type	Description
Primary Group	Default group assigned to a user. Shown in /etc/passwd
Secondary Group	Additional groups a user belongs to. Shown in /etc/group
Supplementary Group	Groups added using usermod -aG or gpasswd -a.
System Group	Created by system or installed packages (GID < 1000).
User Group	Created by admin for organizing users (GID >= 1000).

PRACTICAL WORKFLOW EXAMPLE

1. Create Group → `sudo groupadd projectteam`
2. Add User to Group → `sudo usermod -aG projectteam jeet`
3. Verify Groups → `groups jeet`
4. Remove User (Optional) → `sudo gpasswd -d jeet projectteam`

GROUP COMMANDS QUICK REFERENCE

Command	Use	Example
groupadd	Create new group	sudo groupadd team1
groupdel	Delete group	sudo groupdel team1
usermod -aG	Add user to group	sudo usermod -aG team1 jeet
gpasswd -a	Add user to group	sudo gpasswd -a jeet team1
gpasswd -d	Remove user from group	sudo gpasswd -d jeet team1
gpasswd	Set group password	sudo gpasswd team1
groups	Show groups	groups jeet
cat /etc/group	View all groups	cat /etc/group

EXAMPLE: REAL OUTPUT

1. cat /etc/group

```
root:x:0:
daemon:x:1:
adm:x:4:
sudo:x:27:jeet
devteam:x:1500:jeet,aman
projectteam:x:1600:
students:x:1700:jeet,raj
```

2. groups jeet

```
user@linux:~$ groups jeet
jeet : jeet sudo devteam
students projectteam
```

User 'jeet' is member of 4 groups. (Primary + Secondary)

IMPORTANT NOTES

- Only root (or user with sudo) can manage groups.
- A user must log out and log in again to apply group changes.
- Primary group cannot be removed from a user.
- Be careful while deleting groups (groupdel).

WARNING
Deleting a group does NOT remove users from the system.
It only removes the group entry.

PRO TIP
Organize users with groups for better security & easy management.

GROUP MANAGEMENT WORKFLOW

Create Group (groupadd) → Add Users (usermod -aG) → Verify Groups (groups) → Set Permissions (Safe Access) → Done!

Manage groups smartly, keep your system organized!

ROOT & SUDO

Linux™

Superuser Access & Privileges in Linux

Security | Terminal | Superuser | Privileges | Admin

1. SUDO

sudo

```
user@linux:~$ sudo apt update
[sudo] password for user:
Hit:1 http://archive.ubuntu.com/ubuntu
Reading package lists... Done
```

- sudo allows a permitted user to execute commands as the superuser (root).

2. SU

su

```
user@linux:~$ su
Password:
root@linux:/home/user#
```

- su (substitute user) is used to switch to another user, by default to root user.

3. SUDO -i

sudo -i

```
user@linux:~$ sudo -i
[sudo] password for user:
root@linux:~#
```

- sudo -i provides a login shell as root, loading root's environment.

4. WHOAMI

whoami

```
user@linux:~$ whoami
user
```

- whoami displays the current username of the logged-in user.

5. ROOT

root

```
root@linux:~# whoami
root
```

- root is the superuser account with unrestricted access to the system.

6. ADMINISTRATOR

Administrator

```
C:\> whoami
administrator
```

- Administrator is the highest privileged account in Windows systems.

7. PRIVILEGES

Privileges

- Install / Remove Software
- Modify System Settings
- Access All Files
- Manage Users
- Configure Network
- And much more...

- Privileged accounts can perform restricted and critical system tasks.

8. SUPERUSER

Superuser

```
root@linux:~# id
uid=0(root) gid=0(root) groups=0(root)
```

- Superuser (root) has UID 0 and full control over the entire system.

QUICK TIP

Use sudo for everyday tasks.
Use su or sudo -i only when needed.

USE RESPONSIBLY

Learn | Secure | Protect | Configure | Manage

With great power (root access) comes great responsibility.
Use it **wisely** and **securely**.

Linux™ PROCESSES COMMANDS

Monitor, Find & Manage Processes Like a Pro

Learn | Practice | Use | Explore | Master

1. ps (SNAPSHOT OF PROCESSES)

Display a snapshot of currently running processes.

Syntax

```
ps [options]
```

Example

```
$ ps
PID TTY          TIME CMD
1234 pts/0    00:00:00 bash
2345 pts/0    00:00:01 nano
3456 pts/0    00:00:00 ps
```

Shows processes of the current user.

2. ps aux (DETAILED LIST)

Show all processes for all users in detail.

Syntax

```
ps aux
```

Example

```
$ ps aux
USER   PID %CPU %MEM  VSZ  RSS TTY  STAT  TIME COMMAND
root     1  0.1  0.3 16980 1024 ?    Ss    0:01 systemd
jeet  1234  0.2  1.1 23456 4520 pts/0 S+   0:00 bash
jeet  2345  1.5  2.3 98760 9236 pts/0 R+   0:02 chrome
```

Displays all processes in system.

3. top (REAL-TIME MONITOR)

Real-time dynamic view of running processes.

Syntax

```
top
```

Example

```
top - 10:15:30 up  1:23,  2 users,  load average: 0.32, 0.45, 0.50
Tasks: 103 total,   1 running, 102 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.1 us,  1.2 sy,  0.0 ni, 95.5 id,  1.1 wa,  0.1 hi,  0.0 si
KiB Mem :  8061924 total,  2046120 free,  2145120 used,  3860684 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  4936120 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 2345 jeet      20   0   98760   9236   3540 R   2.0   0.1   0:02.34 chrome
 1234 jeet      20   0   23456   4520   3100 S   0.3   0.1   0:00.45 bash
    1 root      20   0   16980   1024    620 S   0.0   0.0   0:01.10 systemd
```

Press 'q' to quit.

4. htop (ENHANCED TOP)

Interactive process viewer with easy navigation.

Syntax

```
htop
```

Example

```
CPU[|||||||||||||          15.6%]   Tasks: 102, 241 thr; 1 running
Mem[|||||||||||||||2.15G/7.69G]   Load average: 0.32 0.45 0.50
Swp[               128M/2.00G]   Uptime: 01:23:45

  PID USER      PRI  NI  VIRT   RES   SHR S %CPU %MEM    TIME+  Command
 2345 jeet       20   0 98.7M  9.2M  3.5M R  2.0  0.1  0:02.34 chrome
 1234 jeet       20   0 23.4M  4.5M  3.1M S  0.3  0.1  0:00.45 bash
 3456 jeet       20   0 45.6M  6.1M  4.2M S  0.3  0.1  0:00.67 code
 6789 root       20   0 15.6M  1.2M  0.6M S  0.1  0.0  0:00.22 systemd
```

F1=Help F2=Setup F3=Search F10=Quit

5. pgrep (FIND PROCESS BY NAME)

Search for processes by name and show their PID.

Syntax

```
pgrep [options] pattern
```

Example

```
$ pgrep chrome
2345
5678
```

Useful with -a, -u, -f options.

6. pidof (FIND PID OF PROCESS)

Return PID(s) of the running process.

Syntax

```
pidof [options] name
```

Example

```
$ pidof sshd
1122 3344
```

Works with exact process name.

PROCESS STATES (STAT)

State	Meaning
R	Running or runnable
S	Sleeping (interruptible)
D	Uninterruptible sleep
T	Stopped or traced
Z	Zombie (defunct)
I	Idle kernel thread

COMMON PS COLUMNS

Column	Meaning
PID	Process ID
USER	Owner of the process
%CPU	CPU usage percentage
%MEM	Memory usage percentage
VSZ	Virtual memory size
RSS	Resident memory size
TTY	Terminal
STAT	Process status
START	Start time
TIME	CPU time used
COMMAND	Command name

EXAMPLE WORKFLOW

1. List all processes — `ps aux`
2. Find process by name — `pgrep chrome`
3. Get PID — `2345`
4. Monitor in real-time — `top`
5. Kill process (if needed) — `kill 2345`

TIP

Use pgrep or pidof to find PID, then use `kill <PID>` to stop a process.

USEFUL OPTIONS

Command	Option	Description
ps	-e	Show all processes
ps	-f	Full format listing
ps	-u user	Show processes of specific user
top	-d sec	Refresh delay in seconds
pgrep	-a	Show process name with PID
pgrep	-u	Match uid
pidof	-s	Show single PID only

PRO TIP

Learn to read top/htop output. It helps you find high CPU or Memory using processes fast! 🚀

TOP COMMAND OUTPUT EXPLAINED

Tasks
Total, running, sleeping, stopped and zombie processes.

CPU
Shows how CPU time is used (us, sy, id, wa, hi, si).

Memory
Total, used, free and cache/buffer memory.

Process List
PID, USER, %CPU, %MEM, TIME+ and COMMAND.

Keep your system healthy! Monitor processes regularly. 💪

Linux™ PROCESS CONTROL

Manage & Control Processes in Linux

Manage | Terminal | Control | Commands | Master

1. KILL

kill

```
user@linux:~$ kill 1234
# Sends a signal to process with
Process ID 1234
```

- Terminates a process by Process ID (PID).
- Default signal is SIGTERM (15).

Best For: Stop a specific process using PID.

2. KILLALL

killall

```
user@linux:~$ killall firefox
# Kills all processes by name
(e.g., firefox)
```

- Terminates all processes with the same name.
- Sends SIGTERM by default.

Best For: Close all instances of a program.

3. PKILL

pkill

```
user@linux:~$ pkill -f chrome
# Kills processes matching pattern
(e.g., chrome)
```

- Kills processes based on name or pattern.
- Supports options like -f, -u, -t, etc.

Best For: Kill processes using name, pattern or other filters.

4. JOBS

jobs

```
user@linux:~$ jobs
[1]  Running      vim file.txt
[2]  Stopped      sleep 100
```

- Lists background and stopped jobs.
- Shows job ID and current status.

Best For: See the status of jobs started in the shell.

5. BG

bg

```
user@linux:~$ bg %1
[1]  vim file.txt &
```

- Resumes a stopped job in background.
- Job continues and shell is free.

Best For: Run a stopped job in the background.

6. FG

fg

```
user@linux:~$ fg %1
vim file.txt
(now running in foreground)
```

- Brings a background job to foreground.
- You can interact with the process.

Best For: Work on a background job in the foreground.

7. PROCESS ID (PID)

Process ID

```
user@linux:~$ ps -ef | grep firefox
user   1234   1  0 10:10 ?      00:00:02 firefox
```

- Unique number assigned to each process.
- Used by kill, ps and other commands.

Best For: Identify and manage specific processes.

8. TERMINATE & BACKGROUND

Terminate

Background

- **Terminate:** Stop unnecessary or unresponsive processes.
- **Background:** Run tasks without blocking the terminal.

Best For: Keep your system smooth and efficient.

QUICK TIP

Use kill -9 PID to force kill (SIGKILL).
Use Ctrl + Z to stop a process and Ctrl + C to terminate.

PROCESS SMART, WORK FAST

Commands | Signals | Control | Jobs | Monitor

Understand processes.
Control them wisely.
Stay productive in Linux.

Linux™ SYSTEM INFORMATION

Get System & User Information in Linux

Info | Terminal | Users | Time | Calendar | Monitor

1. UNAME

uname

```
user@linux:~$ uname -a
Linux linuxbox 5.15.0-58-generic #64-Ubuntu
SMP x86_64 GNU/Linux
```

- Displays system kernel name and information.
- Options: -a, -r, -s, -m, -p, -o

Best For: Check kernel and system details.

2. HOSTNAME

hostname

```
user@linux:~$ hostname
linuxbox
```

- Displays the system's hostname.
- Shows the current system name.

Best For: View or set the system hostname.

3. HOSTNAMECTL

hostnamectl

```
user@linux:~$ hostnamectl
Static hostname: linuxbox
      Icon name: computer-vm
        Chassis: vm
```

- Shows or changes the system hostname.
- Displays detailed system hostname info.

Best For: Manage and view hostname configuration.

4. UPTIME

uptime

```
user@linux:~$ uptime
10:25:30 up  2 days,  4:15,  2 users,
load average: 0.15, 0.10, 0.05
```

- Shows how long the system has been running.
- Displays load average and logged-in users.

Best For: Check system performance and uptime.

5. WHO

who

```
user@linux:~$ who
user     pts/0        2024-05-18 10:20 (192.168.1.10)
alice    pts/1        2024-05-18 10:22 (192.168.1.11)
```

- Shows logged-in users.
- Displays terminal, login time and host.

Best For: See who is currently logged in to the system.

6. WHOAMI

whoami

```
user@linux:~$ whoami
user
```

- Displays the current logged-in username.
- Shows the effective user ID name.

Best For: Check the current user you are logged in as.

7. DATE

date

```
user@linux:~$ date
Sat May 18 10:25:30 AM IST 2024
```

- Displays or sets the system date and time.
- Options to format or set date and time.

Best For: View or set system date and time.

8. CAL

cal

```
user@linux:~$ cal
      May 2024
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

- Displays a calendar of the current month.
- Options to view other months/years.

Best For: Quick view of calendar and dates.

QUICK TIP

Know your system, manage it better.
Information is power in Linux!

KNOW YOUR SYSTEM, CONTROL YOUR SERVER

Info | Terminal | Users | Time | Calendar | Monitor

These commands are available on almost all Linux distributions.
Simple, powerful and **essential**!

Linux™ DISK USAGE COMMANDS

Monitor Disk Space, Storage & Memory Usage Like a Pro

Learn | Practice | Use | Explore | Master

1. df (DISK FREE – SUMMARY)

Show disk space usage of file systems.

Syntax

```
df
```

Example

```
$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
udev             4096000         0   4096000   0% /dev
tmpfs             821144      1200    819944   1% /run
/dev/sda1       51147060  19456760  29990300  41% /
tmpfs            4105728         0   4105728   0% /dev/shm
/dev/sda2       10240000    512000   9728000   5% /data
tmpfs             821144         0    821144   0% /run/user/1000
```

Shows disk usage in 1K blocks.

2. df -h (HUMAN READABLE)

Show disk space usage in human readable format.

Syntax

```
df -h
```

Example

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           803M  1.2M  802M   1% /run
/dev/sda1        49G   19G   28G  41% /
tmpfs           3.9G     0  3.9G   0% /dev/shm
/dev/sda2       9.8G  512M  9.3G   5% /data
tmpfs           803M     0  803M   0% /run/user/1000
```

Easy to read sizes (K, M, G, T).

3. du (DISK USAGE – FILES/DIRS)

Show disk usage of files and directories.

Syntax

```
du [options] [file/dir]
```

Example

```
$ du -h
4.0K    ./file1.txt
12K     ./docs
4.0K    ./docs/file2.txt
8.0K    ./docs/file3.log
48K     ./images
48K     .
```

Shows size of each file & directory.

4. du -sh (SUMMARY HUMAN READABLE)

Show total disk usage of a directory in human readable format.

Syntax

```
du -sh [directory]
```

Example

```
$ du -sh /home/user
2.3G    /home/user

$ du -sh .
152M    .
```

Shows total size of directory.

5. free (MEMORY USAGE)

Display used and free memory in the system.

Syntax

```
free [options]
```

Example

```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:           7.7G        2.1G        1.8G        152M        3.8G        5.2G
Swap:          2.0G          0B        2.0G                                7.1G
```

-h option shows in human readable format.

6. free -m (MEMORY IN MB)

Display memory usage in MegaBytes.

Syntax

```
free -m
```

Example

```
$ free -m
              total        used        free      shared  buff/cache   available
Mem:           7856        2142        1830         152        3884        5312
Swap:          2047           0        2047                                7119
```

Shows memory in MB.

COMMAND QUICK REFERENCE

Command	Purpose
df	Show disk usage of file systems
df -h	Show disk usage in human readable format
df -T	Show filesystem type
du	Show disk usage of files/directories
du -sh	Show total size of directory (human readable)
du -h --max-depth=1	Show size of subdirectories (summary)
free	Show memory usage
free -h	Show memory usage (human readable)
free -m	Show memory usage in MB
lsblk	List block devices
mount	Show mounted file systems

UNDERSTANDING df OUTPUT

Column	Meaning
Filesystem	Name of file system
Size	Total size
Used	Used space
Avail	Available space
Use%	Percentage of used space
Mounted on	Mount point

UNDERSTANDING du OUTPUT

Example	Meaning
4.0K	Size of file/directory
12K	Size of directory
48K	Total size of directory
.	Current directory total size

COMMON OPTIONS

Option	Description
-h	Human readable format
-m	Show sizes in MB
-g	Show sizes in GB
-s	Summary (only total)
-l	Count symlinks
--max-depth=N	Limit depth of directories

DISK SPACE USAGE EXPLANATION

- **Free Space** — Available space on disk.
- **Used Space** — Space used by files and programs.
- **Buffers/Cache** — Used for system cache (can be freed if needed).
- **Other Usage** — System, logs, metadata, and reserved space.

TIPS & BEST PRACTICES

- Use df -h to quickly check disk space.
- Use du -sh to find large directories.
- Use du -h --max-depth=1 /path to find biggest folders.
- Monitor memory with free -h regularly.
- Clean temporary files and cache to free space.

WARNING
Running out of disk space can cause system errors and service failures.

DISK USAGE WORKFLOW

1. **Check Disk Space** — `df -h`
2. **Find Large Directories** — `du -sh /path`
3. **Analyze Usage** — `du -h --max-depth=1`
4. **Check Memory** — `free -h`
5. **Clean & Optimize** — Remove unnecessary files & cache
6. **Monitor Regularly** — Keep system healthy!

Keep an eye on your storage and memory for a fast and stable system!

Linux™ DISK & BLOCK DEVICES

Manage Disks, Partitions & Block Devices in Linux

Storage | Terminal | Devices | Partitions | Safety | Master

1. LSBLK

lsblk

```
user@linux:~$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda       8:0    0 931.5G  0 disk
├─sda1    8:1    0   100M  0 part /boot/efi
├─sda2    8:2    0    16G  0 part /
└─sda3    8:3    0 915.4G  0 part /home
sdb       8:16   1  14.9G  0 disk
```

- Lists all block devices and their partitions in a tree.
- Shows size, type, mount point and more info.

Best For: Quick overview of disks and partition layout.

2. BLKID

blkid

```
user@linux:~$ blkid
/dev/sda1:  UUID="F1D2-E3C4" TYPE="vfat"
/dev/sda2:  UUID="a1b2c3d4-e5f6-47ab-
            TYPE="ext4" PARTUUID="1111-2222"
/dev/sda3:  UUID="9f8e7d6c-5b4a-3210-
            TYPE="ext4"
```

- Displays block device attributes and filesystem UUID.
- Useful for identifying filesystems and volumes.

Best For: Identify filesystem type, UUID and device details.

3. FDISK

fdisk

```
user@linux:~$ sudo fdisk -l
Disk /dev/sda: 931.5 GiB, 1000204886016 bytes
Disk model: ST1000DM010-2EP102
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512/4096 bytes
Disklabel type: dos
Disk identifier: 0x1a2b3c4d
Device     Boot  Start      End      Sectors  Size Id Type
/dev/sda1  *      2048   206847       204800  100M  EFI System
/dev/sda2       206848 33558527     33351680   16G  Linux
/dev/sda3     33558528 1953523711 1919965184 915.4G Linux
```

- Classic partitioning tool for MBR/GPT disks.
- Create, delete, list and modify partitions.

Best For: Advanced partition management using command-line.

4. PARTED

parted

```
user@linux:~$ sudo parted -l
Model:  ATA ST1000DM010-2EP1 (scsi)
Disk /dev/sda:  1000GB
Sector size (logical/physical): 512B/4096B
Partition Table: gpt
Disk Flags:
Number  Start   End     Size    File system  Name  Flags
1       1049kB  107MB   106MB   fat32        EFI   boot, esp
2       107MB   17.2GB  17.1GB  ext4
3       17.2GB  1000GB  982GB   ext4
```

- Powerful partition editor supporting GPT & MBR.
- More features than fdisk with friendly output.

Best For: Advanced users who need more control and features.

5. DISK

Disk

- A physical storage device that stores data.
- Examples: HDD, SSD, NVMe, USB Drive.
- Represented as /dev/sda, /dev/sdb, etc.

Best For: Understanding physical storage devices in the system.

6. PARTITION

Partition

- A section of a disk used to store data.
- Allows multiple filesystems on one disk.
- Represented as /dev/sda1, /dev/sda2, etc.

Best For: Organizing and managing disk space efficiently.

7. BLOCK DEVICE

Block Device

- Devices that handle data in blocks.
- Examples: Hard drives, SSDs, USB drives.
- Accessed using filesystem or raw mode.

Best For: Working with storage at low level in Linux.

8. DEVICE

Device

- Any hardware or virtual component.
- Includes block, character devices and more.
- Managed under /dev in Linux.

Best For: Identifying and using system devices.

QUICK TIP

Always backup important data before modifying partitions. One wrong command can erase data! ⚠

SMART STORAGE, SAFE SYSTEM

List Devices | Identify FS | Partition | Protect Data | Backup | Monitor Health

Understand your disks.
Manage partitions wisely.
Keep your data safe & secure.

Linux™ MOUNT & UNMOUNT COMMANDS

Manage File System Mounting Like a Pro

Learn | Practice | Use | Explore | Master

1. mount (MOUNT FILE SYSTEM)

Mount a filesystem to a directory (mount point).

Syntax

```
mount [options] <device> <mount_point>
```

Example

```
# Mount a USB drive to /mnt/usb
$ sudo mount /dev/sdb1 /mnt/usb
```

Makes files on the device accessible at the mount point.

2. umount (UNMOUNT FILE SYSTEM)

Unmount a filesystem from a mount point.

Syntax

```
umount [options] <mount_point>
```

Example

```
# Unmount the USB drive
$ sudo umount /mnt/usb
```

Safely removes the filesystem from the mount point.

3. /etc/fstab (FILE SYSTEM TABLE)

Configuration file for automatic mounting at boot time.

Syntax

```
/etc/fstab
```

Example Entry in /etc/fstab

```
# <device>        <mount_point>  <type>  <options>  <dump>  <pass>
UUID=3f4e-8c2a    /mnt/usb       ext4    defaults   0       2
```

Defines filesystems to mount automatically when the system boots.

COMMON OPTIONS

Option	Description
-t <type>	Specify filesystem type (ext4, vfat, ntfs, etc.)
-o <options>	Mount options (rw, ro, defaults, noexec, etc.)
-r	Mount as read-only
-w	Mount as read-write
-a	Mount all filesystems in /etc/fstab
-v	Verbose output

KEY CONCEPTS

DISK — Physical storage device (HDD / SSD) — /dev/sda, /dev/sdb

→ **USB DRIVE** — Removable storage device — /dev/sdb1, /dev/sdc1

→ **FILE SYSTEM** — Organization of data on a device — ext4, vfat, ntfs, xfs

→ **MOUNT POINT** — Directory where the filesystem is mounted — /mnt/usb, /media/usb

→ **STORAGE** — Where your data is stored and accessed — Accessible after mount

CHECK MOUNTED FILE SYSTEMS

1. Using mount Command

```
$ mount
/dev/sda1 on / type ext4 (rw,relatime)
/dev/sdb1 on /mnt/usb type vfat (rw,nosuid,nodev,uid=1000,gid=1000,fmask=0022,dmask=0022)
```

2. Using df -h Command

```
$ df -h
Filesystem      Size  Used  Avail  Use%  Mounted on
/dev/sda1        50G   20G    28G   42%  /
/dev/sdb1        15G  4.0G    11G   27%  /mnt/usb
```

PRACTICAL EXAMPLES

# Create mount point directory	sudo mkdir -p /mnt/usb	Create a directory to use as mount point
# Mount a USB drive (vfat)	sudo mount -t vfat /dev/sdb1 /mnt/usb	Mount vfat (FAT32) filesystem
# Mount an ext4 partition	sudo mount -t ext4 /dev/sdc1 /mnt/data	Mount ext4 filesystem
# Mount all filesystems from fstab	sudo mount -a	Mount all filesystems listed in /etc/fstab
# Unmount a filesystem	sudo umount /mnt/usb	Unmount the filesystem
# Unmount by device	sudo umount /dev/sdb1	Unmount using device name

/etc/fstab EXAMPLE EXPLAINED

Field	Example	Meaning
/dev/sdb1	UUID=3f4e-8c2a...	Device (UUID is recommended)
/mnt/usb	/mnt/usb	Mount point directory
vfat	vfat	Filesystem type
defaults	defaults	Mount options
0	0	Dump frequency (usually 0)
2	2	Pass number (fsck order)

COMMON FILE SYSTEM TYPES

Type	Used For
ext4	Linux native (recommended)
vfat	FAT32 / USB Drives
ntfs	Windows NTFS drives
xfs	High performance filesystems
iso9660	CD/DVD (Read-only)

IMPORTANT NOTES

- Always use sudo for mount/umount.
- Unmount before removing USB or external drives.
- Wrong unmount can cause data loss.
- Use UUID in /etc/fstab for reliability.

PRO TIP

Use UUID instead of /dev/sdX in /etc/fstab because /dev/sdX can change.

MOUNT & UNMOUNT WORKFLOW

Identify Device (lsblk / fdisk -l) → Create Mount Point (mkdir /mnt/usb) → Mount Filesystem (mount /dev/sdb1 /mnt/usb) → Verify Mount (df -h / mount) → Unmount Safely (umount /mnt/usb) → Done!

Mount smart, unmount safe, keep your data secure!

Linux™ FILE SYSTEM

Manage Filesystems & Storage in Linux

Files | Terminal | Storage | Mounts | Integrity | Master

1. DF

df

```
user@linux:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        50G   18G   30G  38% /
/dev/sda1       512M   60M  452M  12% /boot/efi
tmpfs           2G    1.2M  2G    1% /dev/shm
```

- Shows disk space usage of filesystems.
- -h for human readable format.

Best For: Check available storage and filesystem usage.

2. DU

du

```
user@linux:~$ du -sh *
4.0K    Documents
12M     Downloads
2.1G    Videos
512M    backup
```

- Estimates file and directory space usage.
- -s for summary, -h for human readable.

Best For: Find which files or directories are using space.

3. MOUNT

mount

```
user@linux:~$ mount
/dev/sda2 on / type ext4 (rw,relatime)
/dev/sda1 on /boot/efi type vfat (rw,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
/dev/sdb1 on /mnt/data type ext4 (rw,relatime)
```

- Mounts a filesystem to a directory.
- Without args, shows all mounted filesystems.

Best For: Mount filesystems and view mount information.

4. FINDMNT

findmnt

```
user@linux:~$ findmnt
TARGET      SOURCE      FSTYPE  OPTIONS
/           /dev/sda2   ext4    rw,relatime
/boot/efi   /dev/sda1   vfat    rw,relatime
/mnt/data   /dev/sda1   ext4    rw,relatime
/home       /dev/sda3   ext4    rw,relatime
```

- Shows mounted filesystems in a tree view.
- Easy to read and filter.

Best For: View and understand mount relationships easily.

5. STAT

stat

```
user@linux:~$ stat file.txt
  File: file.txt
  Size: 1024            Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d      Inode: 131072      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/ user)   Gid: ( 1000/ user)
Access: 2024-05-18 10:22:33.000000000 +0530
Modify: 2024-05-18 10:15:41.000000000 +0530
Change: 2024-05-18 10:15:41.000000000 +0530
 Birth: 2024-05-18 10:15:41.000000000 +0530
```

- Displays detailed file or filesystem status.
- Includes inode, permissions, owner, timestamps.

Best For: Get detailed info about files, inodes and status.

6. FILE SYSTEM

File System

- A structure that organizes and stores files on storage devices.
- Examples: ext4, xfs, btrfs, vfat, ntfs.
- Provides methods to store, retrieve and manage data.

Best For: Organizing and managing data on storage devices.

7. INODE

Inode

- An index node that stores metadata of a file.
- Contains permissions, owner, timestamps, size, and block pointers.
- Each file has a unique inode number.

Best For: Understanding file metadata and filesystem structure.

8. MOUNT POINT

Mount Point

- A directory where a filesystem is mounted and made accessible.
- Acts as the access point to the storage.
- Example: /mnt/data, /home, /var

Best For: Access and use filesystems in the directory tree.

QUICK TIP

Check storage regularly, manage mounts properly, and understand your filesystem structure!

FILESYSTEM SMART, STORAGE SAFE

Explore | Analyze | Mount | Protect | Optimize | Monitor

A good filesystem strategy ensures better performance, security and data reliability.

Linux™ ARCHIVE FILES

Create, View, Extract & Manage Archives in Linux

Archive | Terminal | Backup | Files | Compress | Master

1. TAR

tar

```
user@linux:~$ man tar
tar - an archiving utility
Usage: tar [OPTION]... [FILE]...
Example: tar -cvf archive.tar /home/user
```

- tar is used to create, extract and manage archive (.tar) files.
- It can also compress archives with gzip, bzip2, xz and more.

Best For: Creating and managing archives of files and directories.

2. TAR -CVF (CREATE)

tar -cvf

```
user@linux:~$ tar -cvf backup.tar /home/user
/home/user/
/home/user/file1.txt
/home/user/docs/
/home/user/docs/report.pdf
...
```

- Creates a new tar archive.
- c: create, v: verbose, f: file name

Best For: Creating backups or archives of files and directories.

3. TAR -XVF (EXTRACT)

tar -xvf

```
user@linux:~$ tar -xvf backup.tar
home/user/
home/user/file1.txt
home/user/docs/
home/user/docs/report.pdf
...
```

- Extracts files from an archive.
- x: extract, v: verbose, f: file name

Best For: Extracting files from existing archives.

4. TAR -TVF (LIST)

tar -tvf

```
user@linux:~$ tar -tvf backup.tar
drwx-xr-x user/user      0 May 18 10:00 home/user/
-rw---r-- user/user   1234 May 18 10:00 home/user/file1.txt
drwx-xr-x user/user      0 May 18 10:00 home/user/docs/
-rwr--r-- user/user   5678 May 18 10:00 home/user/docs/report.pdf
...
```

- Lists the contents of an archive without extracting.
- t: list, v: verbose, f: file name

Best For: Viewing contents of an archive before extraction.

5. ARCHIVE

Archive

- An archive is a collection of files and directories stored in a single file.
- Helps in organization, portability and backup.
- Common formats: `.tar`, `.tar.gz`, `.tar.bz2`, `.tar.xz`

Best For: Storing multiple files in a single archive file.

6. EXTRACT

Extract

- Extracting restores files from an archive to their original state.
- Keeps folder structure and permissions intact.
- Use `tar -xvf archive.tar` to extract.

Best For: Restoring files from backups or shared archives.

7. BACKUP

Backup

- Archives are widely used for backups.
- Easy to store, transfer and restore.
- Supports compression for saving space.
- Example: `tar -czvf backup.tar.gz /data`

Best For: Creating reliable backups of important data.

8. FILES

Files

- Files and directories can be added to archives.
- Maintains metadata like permissions, ownership and timestamps.
- Supports regular files, directories, symlinks and more.

Best For: Organizing and archiving files efficiently.

QUICK TIP

Use relative paths while archiving for portability.
Verify archives after creation for data safety!

ARCHIVE SMART, BACKUP SAFE

Create | Extract | List | Compress | Backup | Restore

Backups today,
peace of mind tomorrow.
Stay safe, stay prepared.

Linux™ COMPRESSION COMMANDS

Compress & Decompress Files Like a Pro

Learn | Practice | Use | Explore | Master

1. gzip (COMPRESS FILES)

GZ

Compress files using gzip.

Syntax

```
gzip [options] file
```

Example

```
$ gzip file.txt
# Creates file.txt.gz
```

Compresses a file and removes the original.

2. gunzip (DECOMPRESS FILES)

GZ

Decompress .gz files using gunzip.

Syntax

```
gunzip [options] file.gz
```

Example

```
$ gunzip file.txt.gz
# Restores file.txt
```

Decompresses a .gz file and restores the original file.

3. bzip2 (COMPRESS FILES)

BZ2

Compress files using bzip2.

Syntax

```
bzip2 [options] file
```

Example

```
$ bzip2 file.txt
# Creates file.txt.bz2
```

Compresses a file with better ratio than gzip.

4. bunzip2 (DECOMPRESS FILES)

BZ2

Decompress .bz2 files using bunzip2.

Syntax

```
bunzip2 [options] file.bz2
```

Example

```
$ bunzip2 file.txt.bz2
# Restores file.txt
```

Decompresses a .bz2 file and restores the original file.

5. xz (COMPRESS FILES)

XZ

Compress files using xz (high compression).

Syntax

```
xz [options] file
```

Example

```
$ xz file.txt
# Creates file.txt.xz
```

High compression ratio, slower but saves more space.

6. unxz (DECOMPRESS FILES)

XZ

Decompress .xz files using unxz.

Syntax

```
unxz [options] file.xz
```

Example

```
$ unxz file.txt.xz
# Restores file.txt
```

Decompresses a .xz file and restores the original file.

COMPRESSION COMMANDS COMPARISON

Command	Extension	Compress	Decompress	Speed	Compression Ratio	Best Use
gzip	.gz	gzip file	gunzip file.gz	⚡⚡⚡	Good	General purpose
bzip2	.bz2	bzip2 file	bunzip2 file.bz2	⚡⚡	Better	Better compression
xz	.xz	xz file	unxz file.xz	⚡	Best	Maximum savings

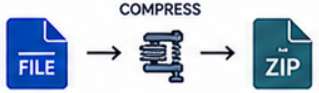
COMMON OPTIONS

Option	Description
-k	Keep original file
-c	Write output to stdout
-d	Decompress
-v	Verbose (show progress)
-1 ... -9	Compression level (1 = fast, 9 = best)
-t	Test compressed file
--best	Best compression (for xz)
--fast	Fast compression (for xz)

EXAMPLES IN ACTION

```
$ ls
file.txt    notes.log    data.csv
$ gzip file.txt          # Compress file.txt -> file.txt.gz
$ ls
file.txt.gz  notes.log   data.csv
$ gunzip file.txt.gz     # Decompress file.txt.gz
$ ls
file.txt    notes.log    data.csv
$ bzip2 notes.log        # -> notes.log.bz2
$ xz   data.csv          # -> data.csv.xz
```

WHAT IS COMPRESSION?



Reduces file size to save storage and bandwidth.

FILE EXTENSIONS

Extension	Created By
.gz	gzip
.bz2	bzip2
.xz	xz

PRACTICAL EXAMPLES

Compress Files

```
$ gzip largefile.txt
$ bzip2 important.log
$ xz  backup.tar
$ xz -9 bigdata.csv
$ gzip -k keepme.txt  # keep original
$ tar -czf archive.tar.gz folder/  # create .tar.gz
```

Decompress Files

```
$ gunzip largefile.txt.gz
$ bunzip2 important.log.bz2
$ unxz backup.tar.xz
$ xz -d bigdata.csv.xz
$ tar -xzf archive.tar.gz   # extract .tar.gz
```

TIPS & BEST PRACTICES

- Use gzip for speed, xz for best compression.
- Always keep important files before compressing.
- Use -k option to keep original files.
- Use -t to test integrity of compressed files.
- For backup, use .tar.gz or .tar.xz.

NOTE

Compressed files take less space, but require extra time to compress or decompress.

PRO TIP

Use compression before uploading or sending files to save time and bandwidth. 🚀

COMPRESSION WORKFLOW

Compress smart, store safe, save space! 💾

ZIP FILES

Linux™

Create, Compress, Extract & Manage ZIP Archives in Linux

Archive | Terminal | Compress | Extract | Directory | Master

1. ZIP

zip

```
user@linux:~$ zip archive.zip file1.txt file2.txt
  adding: file1.txt (deflated 70%)
  adding: file2.txt (deflated 65%)
```

- Creates a ZIP archive.
- Compresses files and directories.

Best For: Creating compressed archives of files and directories.

2. UNZIP

unzip

```
user@linux:~$ unzip archive.zip
Archive:  archive.zip
  inflating: file1.txt
  inflating: file2.txt
```

- Extracts files and directories from a ZIP archive.
- Restores files to the current directory.

Best For: Extracting files from ZIP archives.

3. ZIP -R (RECURSIVE)

zip -r

```
user@linux:~$ zip -r project.zip my_project/
  adding: my_project/ (stored 0%)
  adding: my_project/file1.txt (deflated 68%)
  adding: my_project/docs/ (stored 0%)
  ..
```

- Creates a ZIP archive recursively.
- Includes all files and subdirectories.

Best For: Archiving entire directories and their contents.

4. UNZIP -L (LIST)

unzip -l

```
user@linux:~$ unzip -l archive.zip
Archive:  archive.zip
 Length      Date    Time    Name
--------  ---------- -----   ----------
    1234  05-18-24  10:00   file1.txt
    2345  05-18-24  10:00   file2.txt
--------  ---------- -----   ----------
    3579   2 files
```

- Lists the contents of a ZIP archive.
- Does not extract files.

Best For: Viewing archive contents before extraction.

5. COMPRESS

Compress

- Reduces file size for storage and transfer.
- Saves space and bandwidth.
- Uses ZIP format for compression.

Best For: Efficient storage and sharing of files.

6. EXTRACT

Extract

- Extracts and restores original files.
- Maintains directory structure.
- Preserves file permissions and timestamps (where supported).

Best For: Restoring files from ZIP archives.

7. ARCHIVE

Archive

- An archive is a collection of files compressed into a single file.
- Easy to backup, move and manage.
- Supports encryption (with extra tools).

Best For: Backup, distribution and long-term storage.

8. DIRECTORY

Directory

- Directories can be added to ZIP archives.
- Use -r option to include subdirectories.
- Keeps the folder structure intact.

Best For: Archiving complete directories with all subfolders.

QUICK TIP

Use -r to include directories.
Always check contents with unzip -l before extracting.

ZIP SMART, COMPRESS FAST, EXTRACT EASY

zip — Compress | unzip — Extract | zip -r — Archive Directory | unzip -l — List Contents | Safe & Reliable

ZIP is portable, fast and widely supported.
Keep your data compressed, organized and easy to share.

Linux™ NETWORK INFORMATION COMMANDS

View & Manage Network Configuration Like a Pro

Learn | Practice | Use | Explore | Master

1. ip (IP UTILITY – OVERVIEW)

Show or manipulate routing, devices, policy routing and tunnels.

Syntax

```
ip [options] { object } { command }
```

Example

```
$ ip -V
ip utility, iproute2-6.5.0
```

Modern tool to manage all networking settings.

2. ip addr (IP ADDRESS)

Show or manage IP addresses on network interfaces.

Syntax

```
ip addr [show|add|del]
```

Example

```
$ ip addr show
# Shows all IP addresses
```

Use 'ip addr add <ip>/<mask> dev <iface>' to add a new IP address.

3. ip link (LINK / INTERFACE)

Show or manage network interfaces (link layer).

Syntax

```
ip link [show|set] [dev <iface>]
```

Example

```
$ ip link show
# Shows all network interfaces
```

Use 'ip link set <iface> up/down' to enable or disable interface.

4. ip route (ROUTING TABLE)

Show or manipulate the IP routing table.

Syntax

```
ip route [show|add|del|flush]
```

Example

```
$ ip route show
# Shows all routes
```

Use 'ip route add default via <gw>' to add default gateway.

5. hostname (HOSTNAME)

Show or set the system's host name.

Syntax

```
hostname [ctl] [set-hostname <name>]
```

Example

```
$ hostname
codesardar-pc
$ hostnamectl status
```

Use 'hostnamectl set-hostname <name>' to change hostname.

6. hostname -I (IP QUICK VIEW)

Quickly show all IP addresses assigned to the system.

Syntax

```
hostname -I
```

Example

```
$ hostname -I
192.168.1.105 10.0.0.5
```

Useful for scripts and quick IP lookup.

COMMAND OVERVIEW TABLE

Command	Purpose	What It Shows	Example
ip	IP utility overview	General IP information	ip -V
ip addr	IP addresses	IP addresses of interfaces	ip addr show
ip link	Network interfaces	Status of network interfaces	ip link show
ip route	Routing table	Routes, gateway, destination	ip route show
hostname	Hostname	System hostname	hostname
hostname -I	IP addresses (quick)	All IPs (IPv4/IPv6)	hostname -I

SAMPLE OUTPUTS

```
$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
   inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
   inet 192.168.1.105/24 brd 192.168.1.255 scope global enp0s3
   inet6 fe80::a00:27ff:fe4e:66a1/64 scope link
```

```
$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
   UNKNOWN mode DEFAULT
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
```

```
$ ip route show
default via 192.168.1.1 dev enp0s3 proto dhcp metric 100
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.105
10.0.0.0/24 dev enp0s8 proto kernel scope link src 10.0.0.5
```

```
$ hostname
codesardar-pc
$ hostname -I
 192.168.1.105 10.0.0.5
```

NETWORK INFORMATION EXPLAINED

Network	The connection between devices that allows communication.
IP Address	Unique address assigned to a device on a network (IPv4 / IPv6).
Gateway	The gateway is the router that connects your network to other networks / internet.
Interface	A network interface is the point where your system connects to the network (like enp0s3, wlan0).

COMMON INTERNETWORK INTERFACES

Interface Name	Type / Example
lo	Loopback interface (127.0.0.1)
enp0s3	Wired Ethernet (Common in VirtualBox)
ens33 / eth0	Wired Ethernet (Common in many systems)
wlan0	Wi-Fi Interface
enp0s8	Second Ethernet Adapter (Host-only / NAT)

COMMON ROUTES EXPLAINED

Route	Meaning
default via <gateway>	Default gateway for internet access
<network>/mask	Route to a specific network
dev <iface>	Use this interface to reach destination
src <ip>	Source IP address used for this route
metric <num>	Priority (lower number = higher priority)

PRACTICAL EXAMPLES

```
# Show all IP addresses
$ ip addr

# Bring interface down
$ ip link set enp0s3 down

# Bring interface up
$ ip link set enp0s3 up

# Add default gateway
$ ip route add default via 192.168.1.1

# Check routing table
$ ip route show

# Change hostname
$ hostnamectl set-hostname myserver
```

IMPORTANT NOTES

- Use 'ip' command (not ifconfig) – it is modern and more powerful.
- Changes using ip route or ip addr are temporary. Use /etc/network/interfaces or NetworkManager for permanent settings.
- Run commands with sudo if permission denied.
- Wrong route or interface settings can break your network connection.

PRO TIP

Learn 'ip' command well – it is a must-have skill for Linux system administrators!

NETWORK INFORMATION WORKFLOW

1. Check IP Addresses (ip addr)
2. Check Interfaces (ip link)
3. Check Routes (ip route)
4. Verify Gateway (Default Route)
5. Confirm Connectivity (Ping / Test)

Know your network, control your system!

Linux™ NETWORK TESTING

Test Connectivity, DNS & Routes in Linux

Network Test | Terminal | DNS | Route | Connectivity | Master

1. PING

ping

```
user@linux:~$ ping -c 4 google.com
PING google.com (142.250.72.100) 56(84) bytes
64 bytes from 142.250.72.100: icmp_seq=1 ttl=117 time=11.6 ms
64 bytes from 142.250.72.100: icmp_seq=2 ttl=117 time=11.4 ms
64 bytes from 142.250.72.100: icmp_seq=3 ttl=117 time=11.3 ms
64 bytes from 142.250.72.100: icmp_seq=4 ttl=117 time=11.2 ms

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss,
round-trip min/avg/max/mdev = 11.2/11.3/11.6/0.2 ms
```

- Tests basic network connectivity to a host.
- Measures latency and packet loss.

Best For: Quick connectivity and latency testing.

2. TRACEROUTE

traceroute

```
user@linux:~$ traceroute google.com
traceroute to google.com (142.250.72.100), 30 hops max
1   192.168.1.1      1.192 ms   1.256 ms   1.312 ms
2   10.0.0.1         5.123 ms   5.290 ms   5.412 ms
3   203.0.113.1      8.452 ms   8.621 ms   8.719 ms
4   203.0.113.5     12.275 ms  12.498 ms  12.612 ms
5   142.250.72.100  11.147 ms  11.202 ms  11.266 ms
```

- Traces the route packets take to a destination.
- Shows each hop and response time.

Best For: Diagnosing network route and hop-by-hop delays.

3. TRACEPATH

tracepath

```
user@linux:~$ tracepath google.com
 1?: [LOCALHOST]
 1:  192.168.1.1                    0.789ms
 1:  10.0.0.1                       3.456ms
 1:  203.0.113.1                    6.789ms
 1:  203.0.113.5                   10.542ms
 1:  142.250.72.100                11.320ms
Resume: pmtu 1500 hops 5 back 5
```

- Like traceroute but uses TCP/UDP packets.
- Finds MTU and avoids unnecessary requests.

Best For: Path discovery and MTU analysis.

4. HOST

host

```
user@linux:~$ host google.com
google.com has address 142.250.72.100
google.com has IPv6 address 2404:6800:4002:819::2004
google.com mail is handled by 10 smtp.google.com.
google.com mail is handled by 20 alt1.gmail-smtp-in.l.google.com.
```

- Performs DNS lookup for domain or IP.
- Shows A, AAAA, MX and other DNS records.

Best For: Quick DNS record information lookup.

5. NSLOOKUP

nslookup

```
user@linux:~$ nslookup google.com
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   google.com
Address:  142.250.72.100
Name:   google.com
Address:  2404:6800:4002:819::2004
```

- Interactive DNS query tool.
- Supports domain, record type and reverse lookup.

Best For: Advanced DNS queries and troubleshooting.

6. NETWORK TEST

Network Test

- Verifies if a host is reachable.
- Checks response time and packet loss.
- Useful for monitoring network health.

Best For: Basic connectivity verification.

7. DNS

DNS

- Domain Name System translates names to IP addresses.
- Critical for accessing websites and services.
- Includes A, AAAA, MX, NS, CNAME records.

Best For: Name resolution and DNS troubleshooting.

8. ROUTE

Route

- Shows the path packets take to reach destination.
- Helps identify network hops and delays.
- Useful for diagnosing routing issues.

Best For: Finding network path and routing issues.

9. CONNECTIVITY

Connectivity

- Ensures hosts and services are reachable.
- Helps detect network failures early.
- Essential for stable network operations.

Best For: Ensuring stable and reliable network access.

10. SUMMARY

Summary

- These tools help test and diagnose networks.
- Identify issues in connectivity, DNS and routes.
- Improve troubleshooting and performance.

Best For: Network administrators, developers and sysadmins.

QUICK TIP

Use these tools together for effective network troubleshooting.
Start with ping, then traceroute or tracepath! 🚀

TEST, ANALYZE & IMPROVE YOUR NETWORK

ping | traceroute | tracepath | host | nslookup | DNS | Route | Connectivity

A tested network is a reliable network.
Test, Monitor, Optimize & Stay Connected Always!

Linux™ CURL COMMANDS

Transfer Data With URLs Like a Pro

Learn | Practice | Use | Explore | Master

1. curl (BASIC REQUEST)

Send a GET request to a URL and show the response.

Syntax

```
curl [options] <URL>
```

Example

```
$ curl https://example.com
<!doctype html>
<html>...</html>
```

Fetches the content of the URL.

2. curl -I (HEADERS ONLY)

Fetch only the HTTP headers from the server.

Syntax

```
curl -I <URL>
```

Example

```
$ curl -I https://example.com
HTTP/2 200
Date: Tue, 20 May 2025 10:00:00 GMT
Content-Type: text/html
Server: nginx
...
```

Useful to check status, type, size, etc.

3. curl -X GET (EXPLICIT GET)

GET — Send a GET request explicitly using -X GET.

Syntax

```
curl -X GET <URL>
```

Example

```
$ curl -X GET "https://api.example.com/data"
{
  "status": "success",
  "data": [1, 2, 3, 4]
}
```

Same as default, but more explicit.

4. curl -X POST (SEND DATA)

POST — Send a POST request with data to a server.

Syntax

```
curl -X POST <URL> -d "key=value"
```

Example

```
$ curl -X POST https://api.example.com/login \
  -d "username=admin&password=1234"
{
  "status": "logged_in",
  "token": "abc123xyz"
}
```

Sends data in the body of POST request.

5. COMMON OPTIONS

Option	Description
-I	Fetch headers only
-X <METHOD>	Specify request method (GET, POST, PUT, DELETE)
-d <DATA>	Send data in POST request
-H "Header: value"	Add custom header
-o <file>	Save output to a file
-s	Silent mode (no progress)
-L	Follow redirects
-v	Verbose output

HTTP STATUS CODES (COMMON)

Code	Meaning	Description
200	OK	Request successful
201	Created	Resource created
301	Moved Permanently	Resource moved
400	Bad Request	Invalid request
401	Unauthorized	Authentication required
403	Forbidden	Access denied
404	Not Found	Resource not found
500	Internal Server Error	Server-side error

CURL WORKFLOW

1. Enter URL (Where to send request)
2. Choose Method (GET / POST / etc.)
3. Add Options (Headers, Data, Parameters)
4. Send Request (curl executes the request)
5. Get Response (Server sends data back)

TIP: Use -v for detailed info, -s for silent mode, and -L to follow redirects.

PRACTICAL EXAMPLES

```
$ curl https://example.com                          # Get webpage content
$ curl -I https://example.com                       # Get headers only
$ curl -X GET https://api.example.com/data          # Explicit GET request
$ curl -X POST https://api.example.com/login \
  -d "user=admin&pass=1234"                         # POST with data
$ curl -H "Authorization: Bearer token123" \
  https://api.example.com/protected                 # GET with header
$ curl -o output.html https://example.com           # Save output to file
$ curl -s https://example.com                       # Silent mode
$ curl -L https://bit.ly/example                    # Follow redirects
```

KEY CONCEPTS

- **URL** — The address of the resource.
- **HTTP** — Protocol used to communicate.
- **REQUEST** — What you send to the server.
- **RESPONSE** — What server sends back.
- **API** — Application Programming Interface. Used to get / send data in apps.

USE CASES

- **Test APIs** — Send requests to test API endpoints.
- **Check Websites** — Check status, headers or content of sites.
- **Upload Data** — Send data to server using POST.
- **Download Files** — Download files from URLs easily.
- **Automation** — Use in scripts for automation tasks.

BEST PRACTICES

- Always use HTTPS for secure communication.
- Use -I to check server status before full request.
- Use -v while debugging requests.
- Do not expose sensitive data in URLs.
- Validate API responses in your scripts.

PRO TIP

curl is a powerful tool for DevOps, SysAdmins and Developers. Learn it well, it's super useful! 🚀

CURL IN ACTION

Type curl command → Add options & data → Request goes to the server → Server processes the request → Response comes back to you → Success!

curl helps you connect, test and automate the web! 💪

Linux™ WGET

Powerful Tool for Downloading Files & Websites in Linux

Download | Terminal | URL | File | Resume | Website | Master

1. WGET

wget

```
user@linux:~$ wget https://example.com/file.zip
--2024-05-18 10:20:30--  https://example.com/file.zip
Resolving example.com (example.com)... 93.184.216.34
Connecting to example.com (example.com)|93.184.216.34|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10485760 (10M) [application/zip]
Saving to: 'file.zip'

file.zip      100%[==================>]  10.00M  1.25MB/s   in 8.0s

2024-05-18 10:20:38 (1.25 MB/s) - 'file.zip' saved [10485760/10485760]
```

- Downloads a file from the specified URL.
- Simple and easy to use.

Best For: Quickly downloading files from the internet.

2. WGET -C (RESUME)

wget -c

```
user@linux:~$ wget -c https://example.com/largefile.iso
--2024-05-18 10:25:10--  https://example.com/largefile.iso
Resolving example.com (example.com)... 93.184.216.34
Connecting to example.com (example.com)|93.184.216.34|:443... connected.
HTTP request sent, awaiting response... 206 Partial Content
Length: 2147483648 (2.0G), 1048576000 (1.0G) remaining [application/octet-stream]
Saving to: 'largefile.iso'
..
largefile.iso    50%[=======>          ]  1.00G  1.30MB/s   eta 13m 20s
```

- Resumes an interrupted download.
- Uses existing partial file to continue.

Best For: Resuming large downloads after interruption.

3. WGET -O (OUTPUT)

wget -O

```
user@linux:~$ wget -O ubuntu.iso https://releases.ubuntu.com/ubuntu.iso
--2024-05-18 10:30:45--  https://releases.ubuntu.com/ubuntu.iso
Resolving releases.ubuntu.com (releases.ubuntu.com)... 91.189.88.242
Connecting to releases.ubuntu.com (releases.ubuntu.com)|91.189.88.244|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3145728000 (3.0G) [application/octet-stream]
Saving to: 'ubuntu.iso'

ubuntu.iso      100%[==================>]  3.00G  2.10MB/s   in 24m 15s

2024-05-18 10:55:00 (2.10 MB/s) - 'ubuntu.iso' saved [3145728000/3145728000]
```

- Saves the downloaded file with a specific name.
- Default filename from URL can be overridden.

Best For: Saving files with custom names.

4. WGET -R (RECURSIVE)

wget -r

```
user@linux:~$ wget -r -np -k https://example.com
--2024-05-18 11:00:10--  https://example.com/
Resolving example.com (example.com)... 93.184.216.34
Connecting to example.com (example.com)|93.184.216.34|:443... connected.
HTTP request sent, awaiting response... 200 OK
Saving to: 'example.com/index.html'
...
FINISHED --2024-05-18 11:01:05--
Total wall clock time: 55s
Downloaded: 124 files, 12.4M in 54s (234 KB/s)
```

- Downloads a whole website recursively.
- Useful for offline browsing.

Best For: Downloading entire websites for offline use.

5. DOWNLOAD

Download

- wget is used to download files from the internet.
- Supports HTTP, HTTPS and FTP protocols.
- Works in background and is reliable.

Best For: Fast and reliable file downloads.

6. URL

URL

- Stands for Uniform Resource Locator.
- Specifies the location of the file or website.
- Example: https://example.com/file.zip

Best For: Providing the source address.

7. FILE

File

- The target file to be downloaded.
- Can be saved with default or custom name.
- Stores data locally in your system.

Best For: Managing downloaded files.

8. WEBSITE

Website

- A collection of related web pages.
- Can be downloaded recursively with wget.
- Useful for offline viewing and backup.

Best For: Archiving websites locally.

QUICK TIP

Use -c to resume, -O to rename, -r to download websites.
Combine options for more powerful downloads!

WGET COMMANDS AT A GLANCE

wget	wget -c	wget -O	wget -r
Download File	Resume Download	Save with Name	Download Website

wget makes downloads simple, reliable and powerful.
Download Smart, Work Smart! 🚀

Linux™ SSH COMMANDS

Access Remote Server Securely Like a Pro

Learn | Practice | Use | Explore | Master

1. ssh (BASIC LOGIN)

Connect to a remote server using SSH.

Syntax

```
ssh [user@]host
```

Example

```
$ ssh user@192.168.1.10
user@192.168.1.10's password:
Welcome to Ubuntu 22.04 LTS!
```

Connects to remote host using default SSH port 22.

2. ssh user@host (EXPLICIT USER)

Connect to remote server with a specific username.

Syntax

```
ssh user@host
```

Example

```
$ ssh admin@server.example.com
admin@server.example.com's password:
Last login: Tue May 20 10:00:00 2025
```

Useful when logging in with a specific user.

3. -p (SPECIFY PORT)

Connect using a different SSH port.

Syntax

```
ssh -p [port] user@host
```

Example

```
$ ssh -p 2222 user@192.168.1.10
user@192.168.1.10's password:
Welcome!
```

Default SSH port is 22.
Use -p to connect on custom port.

4. -i (IDENTITY FILE / PRIVATE KEY)

Connect using a private key instead of password.

Syntax

```
ssh -i /path/to/key user@host
```

Example

```
$ ssh -i ~/.ssh/id_rsa user@server.com
Welcome to server.com!
```

More secure than password login.
Make sure key has correct permissions.

5. COMMON OPTIONS

Option	Description
-p [port]	Specify SSH port
-i [key]	Use identity file (private key)
-v	Verbose mode (debug info)
-C	Enable compression
-L [port:host:hostport]	Local port forwarding
-N	Do not execute remote command
-T	Disable pseudo-terminal allocation

HOW SSH WORKS

- SSH encrypts all data between client and server.
- Authentication can be done using password or private key.
- Once connected, you get a secure remote terminal.

PRACTICAL EXAMPLES

Use Case	Command Example
Basic login	ssh user@192.168.1.10
Login with custom port	ssh -p 2222 user@192.168.1.10
Login with private key	ssh -i ~/.ssh/id_rsa user@server.com
Verbose connection	ssh -v user@server.com
Disable terminal (run command)	ssh user@server.com "uptime"
Local port forwarding	ssh -L 8080:localhost:80 user@server.com

SSH CONFIG FILE (~/.ssh/config)

Simplify SSH connections using config file.

Example:

```
Host myserver
    HostName 192.168.1.10
    User admin
    Port 2222
    IdentityFile ~/.ssh/id_rsa
```

Now connect simply with:

```
$ ssh myserver
```

KEY MANAGEMENT COMMANDS

Command	Description
ssh-keygen	Generate new SSH key pair
ssh-copy-id user@host	Copy public key to server
ssh-add ~/.ssh/id_rsa	Add private key to agent
ssh-keyscan host	Scan host and add to known_hosts

NOTE

- Never share your private key.
- Use strong passphrase for your key.
- Keep your system and SSH server updated.

SSH CONNECTION TYPES

Password Authentication
Login using username & password.

Key based Authentication
Login using private key (no password).

Port Forwarding
Securely forward ports to access services.

Tunneling
Create encrypted tunnel for network traffic.

COMMON SSH FILES

File	Purpose
~/.ssh/id_rsa	Private key
~/.ssh/id_rsa.pub	Public key
~/.ssh/authorized_keys	Authorized public keys (on server)
~/.ssh/known_hosts	Stores known hosts fingerprints
/etc/ssh/sshd_config	SSH server configuration

TROUBLESHOOTING TIPS

- Connection refused → Check SSH service on server.
- Permission denied → Check username or key permissions.
- Host key verification failed → Clear known_hosts entry.
- Use ssh -v for detailed debug information.
- Ensure port is open in firewall/router.

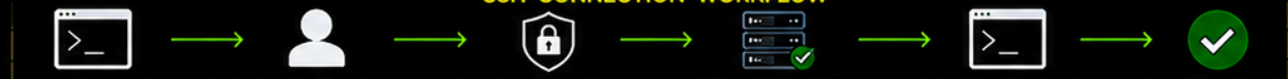
QUICK REFERENCE

```
ssh user@host
ssh -p 2222 user@host
ssh -i ~/.ssh/id_rsa user@host
ssh -L 8080:localhost:80 user@host
ssh -v user@host
```

PRO TIP

Use key-based authentication for better security and easier automation!

SSH CONNECTION WORKFLOW



SSH keeps your connection secure, encrypted and protected!

Linux™ SCP (SECURE COPY) COMMANDS

Securely Copy Files Between Local & Remote Servers

Learn | Practice | Use | Explore | Master

1. scp (BASIC SYNTAX)

Securely copy files between local and remote systems.

Syntax

```
scp [options] source destination
```

Example

```
$ scp file.txt user@192.168.1.10:/home/user/file.txt
```

SCP uses SSH to transfer files securely.

2. scp file user@host: (COPY TO REMOTE)

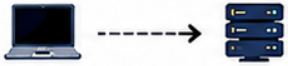
Copy file from local machine to remote server.

Syntax

```
scp file user@host:/path/
```

Example

```
$ scp report.pdf user@server.com:/home/user/
```



3. scp user@host:file . (COPY FROM REMOTE)

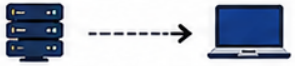
Copy file from remote server to local machine.

Syntax

```
scp user@host:/path/file .
```

Example

```
$ scp user@server.com:/home/user/data.txt .
```



4. scp -r (RECURSIVE COPY)

Copy directories recursively with all files.

Syntax

```
scp -r folder user@host:/path/
```

Example

```
$ scp -r myfolder user@192.168.1.10:/backup/
```

Use -r to copy entire folders with content.

COMMON SCP OPTIONS

Option	Description
-r	Copy directories recursively
-P [port]	Specify custom SSH port
-i [key]	Use identity file (private key)
-p	Preserve file modification times
-C	Enable compression
-v	Verbose mode (show details)
-q	Quiet mode (less output)

SCP SOURCE & DESTINATION EXAMPLES

Source	Destination	Example Command
file.txt	user@host:/path/	Local file → Remote: `scp file.txt user@host:/path/`
user@host:/file.txt	.	Remote file → Local: `scp user@host:/file.txt .`
user@host:/file.txt	user2@host2:/path/	Remote → Remote: `scp user@h1:/file user2@h2:/path/`
-r folder/	user@host:/backup/	Folder → Remote: `scp -r folder/ user@host:/backup/`

PRACTICAL EXAMPLES

```
$ scp file.txt user@192.168.1.10:/home/user/        # Copy file to remote
$ scp user@192.168.1.10:/home/user/file.txt .       # Copy file from remote
$ scp -r project/ user@server.com:/var/www/         # Copy folder to remote
$ scp -P 2222 file.txt user@host:/path/             # Use custom port 2222
$ scp -i key.pem file.txt user@host:/path/          # Use private key
$ scp -r user@host:/data/backup .                   # Download folder
$ scp -v file.txt user@host:/tmp/                   # Verbose mode
```

COMMON USE CASES

- Backup important files to remote server
- Deploy website files to live server
- Transfer logs and reports securely
- Share files between servers
- Automate file transfer in scripts

SECURITY TIPS

- Always use SSH key (-i key.pem) instead of password.
- Use custom port to increase security (-P 2222).
- Do not share your private key.
- Enable compression for large files (-C for faster transfer).

SCP VS CP (DIFFERENCE)

cp (Local Copy)	scp (Secure Copy)
Works only on same system	Works between local & remote
Faster but not secure	Secure (uses SSH)
Example: cp file.txt /home/user/	Example: cp file user@host:/path/

VS

SCP FILE PERMISSIONS & INFO

Concept	Command / Description
Preserve permissions	scp -p file user@host:/path/
Show verbose details	scp -v file user@host:/path/
Compress for speed	scp -C largefile user@host:/path/
Use identity key	scp -i key.pem file user@host:/path/

COMMON FILE PATH EXAMPLES

- /home/user/ → User Home Directory
- /var/www/html/ → Web Server Files
- /tmp/ → Temporary Files
- /etc/ → System Configuration
- /backup/ → Backup Directory

QUICK REFERENCE

```
$ scp file user@host:/path/      → Upload File
$ scp user@host:/file .          → Download File
$ scp -r folder user@host:/      → Upload Folder
$ scp -P 2222 file user@host:/   → Custom Port
$ scp -i key.pem file user@host:/ → With Key
```

PRO TIP

SCP is your best friend when you need to transfer files securely and quickly!

SCP WORKFLOW (IN SIMPLE STEPS)

Secure, Fast & Reliable File Transfer with SCP!

Linux™ SFTP (FILE TRANSFER) COMMANDS

Transfer Files Securely Over SSH Like a Pro

Learn | Practice | Use | Explore | Master

1. sftp (CONNECT)

Connect to a remote server using SFTP.

Syntax

```
sftp user@host
```

Example

```
$ sftp user@192.168.1.10
Connected to 192.168.1.10.
sftp>
```

Starts an interactive SFTP session.

2. put (UPLOAD FILE)

Upload a file from your local machine to remote server.

Syntax

```
put local_file remote_path/
```

Example

```
sftp> put report.pdf /home/user/docs/
Uploading report.pdf to /home/user/docs/
```

Uploads a file to the remote server.

3. get (DOWNLOAD FILE)

Download a file from remote server to your local machine.

Syntax

```
get remote_file local_path/
```

Example

```
sftp> gut /home/user/docs/data.txt ./
Downloading /home/user/docs/data.txt to .
```

Downloads a file from the remote server.

4. lcd (LOCAL CHANGE DIRECTORY)

Change the local working directory.

Syntax

```
lcd local_directory
```

Example

```
sftp> lcd /home/jeet/downloads
Local directory is /home/jeet/downloads
```

Changes directory on your local machine.

5. ls (LIST REMOTE FILES)

List files and directories in the remote directory.

Syntax

```
ls
```

Example

```
sftp> ls
data.txt   report.pdf   docs/   images/
```

Shows files and folders on remote server.

6. cd (CHANGE REMOTE DIRECTORY)

Change the remote working directory.

Syntax

```
cd remote_directory
```

Example

```
sftp> cd /var/www/html
sftp> pwd
Remote working directory: /var/www/html
```

Changes directory on the remote server.

SFTP COMMON COMMANDS QUICK REFERENCE

Command	Description	Example
sftp user@host	Connect to remote server	sftp user@192.168.1.10
put file	Upload file to remote	put file.txt /home/user/
get file	Download file from remote	get /home/user/file.txt ./
lcd path	Change local directory	lcd /home/jeet/downloads
ls	List remote files	ls
cd path	Change remote directory	cd /var/www/html
pwd	Show remote working directory	pwd
rm file	Delete remote file	rm old.txt
mkdir dir	Create remote directory	mkdir new_folder
rmdir dir	Remove remote directory	rmdir empty_folder
bye / exit	Exit SFTP session	bye

SFTP SESSION EXAMPLE

```
$ sftp user@192.168.1.10
Connected to 192.168.1.10.
sftp> pwd
Remote working directory: /home/user
sftp> ls
data.txt    report.pdf    docs/    images/
sftp> put notes.txt /home/user/docs/
Uploading notes.txt to /home/user/docs/notes.txt
sftp> get /home/user/data.txt ./
Downloading /home/user/data.txt to ./data.txt
sftp> lcd /home/jeet/downloads
Local directory is /home/jeet/downloads
sftp> bye
Disconnected from 192.168.1.10.
```

FILE TRANSFER WORKFLOW

Upload (put)

Local File → Remote Server

Download (get)

Remote File → Local Machine

USEFUL SFTP TIPS

- SFTP uses SSH for secure file transfer.
- Always use strong passwords or SSH keys.
- Use `put` to upload and `get` to download files.
- Use `lcd` to change your local directory.
- Use `ls -l` for detailed file information.
- Exit safely with `bye` or `exit` command.

NOTE

- SFTP is more secure than FTP.
- Make sure you have proper permissions on remote files and directories.
- Keep your SSH keys safe and never share them.

FILE TRANSFER EXAMPLES

Upload Multiple Files

```
sftp> put file1.txt /home/user/
sftp> put file2.txt /home/user/
sftp> put file3.txt /home/user/
```

Download Multiple Files

```
sftp> get /home/user/file1.txt ./
sftp> get /home/user/file2.txt ./
sftp> get /home/user/file3.txt ./
```

SFTP COMMANDS CHEAT SHEET

sftp user@host	put file path	get file path	lcd directory	ls	cd directory	pwd	bye / exit
Connect	Upload File	Download File	Change Local Dir	List Remote Files	Change Remote Dir	Print Working Dir	Exit SFTP

PRO TIP

Use SSH Keys instead of passwords for automated and secure transfers.

DNS COMMANDS

Query DNS, Domain, IP Address & Name Server Information

Learn | Practice | Use | Explore | Master

1. dig – DOMAIN INFORMATION GROPER

Powerful tool to query DNS servers and get detailed information about DNS records.

Syntax

```
dig [domain] [record-type] [@dns-server] [options]
```

Example

```
$ dig example.com
; <<>> DiG 9.18.18 <<>> example.com
;; global options: +cmd
;; Got answer:
example.com.        3600    IN    A     93.184.216.34
```

Default record type is A (IPv4 address).

2. nslookup – NAME SERVER LOOKUP

Query DNS to get domain or IP address information using an interactive or non-interactive mode.

Syntax

```
nslookup [domain | IP] [dns-server]
```

Example

```
$ nslookup example.com
Server:     8.8.8.8
Address:    8.8.8.8

Name:       example.com
Address:    93.184.216.34
```

Interactive mode: nslookup (then type commands)

3. host – SIMPLE DNS LOOKUP

Simple command-line tool for performing DNS lookups.

Syntax

```
host [domain] [record-type] [dns-server]
```

Example

```
$ host example.com
example.com has address 93.184.216.34
example.com mail is handled by 10 mail.example.com.
```

Great for quick DNS lookups.

4. COMMON OPTIONS (dig)

Option	Description
@server	Use specific DNS server
-t type	Query specific record type (A, MX, NS, TXT, etc.)
+short	Short output (only the answer)
-x IP	Reverse lookup (IP to domain)
+trace	Trace the query from root to authoritative server
+noall +answer	Show only the answer section
-6	Query IPv6 address (AAAA record)
+time=sec	Set query timeout in seconds

COMMON DNS RECORD TYPES

Type	Meaning	Example
A	IPv4 Address	example.com. A 93.184.216.34
AAAA	IPv6 Address	example.com. AAAA 2001:db8::1
MX	Mail Exchange	example.com. MX 10 mail.example.com.
NS	Name Server	example.com. NS ns1.example.com.
CNAME	Canonical Name	www.example.com. CNAME example.com.
TXT	Text Record	example.com. TXT "v=spf1 include:_spf..."
PTR	Pointer (Reverse)	34.216.184.93.in-addr.arpa. PTR example.com.
SOA	Start of Authority	example.com. SOA ns1.example.com. admin.example.com.

PRACTICAL EXAMPLES

Use Case	dig Command	nslookup Command	host Command
Get IP (A record)	dig example.com	nslookup example.com	host example.com
Get IPv6 (AAAA)	dig example.com AAAA	nslookup -type=AAAA example.com	host -t AAAA example.com
Get MX records	dig example.com MX	nslookup -type=MX example.com	host -t MX example.com
Get NS records	dig example.com NS	nslookup -type=NS example.com	host -t NS example.com
Get TXT record	dig example.com TXT	nslookup -type=TXT example.com	host -t TXT example.com
Reverse lookup	dig -x 8.8.8.8	nslookup 8.8.8.8	host 8.8.8.8
Use custom DNS	dig example.com @1.1.1.1	nslookup example.com 1.1.1.1	host example.com 1.1.1.1

DNS QUERY FLOW (dig +trace)

```
$ dig example.com +trace
.               518400  IN  NS  a.root-servers.net.
.com.           172800  IN  NS  a.gtld-servers.net.
example.com.    86400   IN  NS  ns1.example.com.
example.com.    3600    IN  A   93.184.216.34
```

INTERACTIVE NSLOOKUP MODE

```
$ nslookup
> server 8.8.8.8
> example.com
Server:     8.8.8.8
Address:    8.8.8.8

Name:       example.com
Address:    93.184.216.34

> set type=MX
> example.com
mail.example.com    preference = 10

> exit
```

BEST PRACTICES

- Use dig for detailed and advanced DNS queries.
- Use nslookup for quick checks and interactive queries.
- Use host for simple and fast lookups.
- Always specify record type for accurate results.
- Use trusted DNS servers (8.8.8.8, 1.1.1.1) for reliable results.

COMMON PUBLIC DNS SERVERS

DNS Server	Primary IP	Secondary IP
Google DNS	8.8.8.8	8.8.4.4
Cloudflare DNS	1.1.1.1	1.0.0.1
Quad9 DNS	9.9.9.9	149.112.112.112
OpenDNS	208.67.222.222	208.67.220.220

QUICK REFERENCE

dig
Best for advanced and detailed DNS information.

nslookup
Good for quick checks and interactive use.

host
Best for simple and fast DNS lookups.

COMMON DNS RECORDS EXPLAINED

- A : Maps domain to IPv4 address.
- AAAA : Maps domain to IPv6 address.
- MX : Mail servers for the domain.
- NS : Name servers responsible for the domain.
- CNAME : Alias of another domain.
- TXT : Text information (SPF, verification, etc.).

PRO TIP

dig +short example.com gives only the IP address – perfect for scripts and automation!

DNS LOOKUP WORKFLOW

1. Enter DNS Command
2. Send Query to DNS Server
3. DNS Resolves Request
4. Receive DNS Records
5. Display Result (IP / Info)

DNS is the phonebook of the Internet. Use these commands to find answers behind every domain!

Linux™ NETWORK CONNECTIONS COMMANDS

Monitor, Manage & Troubleshoot Network Connections Like a Pro

Learn | Practice | Use | Explore | Master

1. ss – SOCKET STATISTICS

Modern replacement for netstat. Faster and more powerful.

Syntax

```
ss [options]
```

Example

```
$ ss
Netid  State   Recv-Q  Send-Q  Local Address:Port
tcp    ESTAB   0       0       192.168.1.10:22
tcp    ESTAB   0       0       192.168.1.10:80
udp    UNCONN  0       0       0.0.0.0:53
```

Shows active TCP, UDP and raw socket connections.

2. ss -tuln – LISTENING PORTS (TCP/UDP)

Show listening TCP and UDP ports with numeric addresses.

Syntax

```
ss -tuln
```

Example

```
$ ss -tuln
Netid  State    Local Address:Port
tcp    LISTEN   0.0.0.0:22
tcp    LISTEN   0.0.0.0:80
udp    UNCONN   0.0.0.0:53
udp    UNCONN   127.0.0.1:123
```

Shows all listening TCP and UDP ports.

3. netstat – NETWORK STATISTICS

Traditional tool to display network connections, routing, and interface stats.

Syntax

```
netstat [options]
```

Example

```
$ netstat -tuln
Proto Recv-Q Send-Q Local Address   State
tcp   0      0      0.0.0.0:22      LISTEN
tcp   0      0      0.0.0.0:80      LISTEN
udp   0      0      0.0.0.0:53      -
udp   0      0      127.0.0.1:123   -
```

Use `netstat -tuln` to view listening ports.

4. lsof – LIST OPEN FILES

List open files and the network connections opened by processes.

Syntax

```
lsof [options]
```

Example

```
$ lsof -i
COMMAND         PID    USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
sshd            1234   root   3u   IPv4  12345  0t0      TCP *:22 (LISTEN)
nginx           2345 www-data  6u   IPv4  23456  0t0      TCP *:80 (LISTEN)
systemd-resolve 567  systemd-resolve 13u IPv4  34567  0t0      UDP UDP *:53
-
```

Shows which process is using which port or connection.

5. COMMON OPTIONS

Option	Description
-t	Show TCP connections
-u	Show UDP connections
-l	Show only listening sockets
-n	Show numeric addresses and ports
-p	Show process using the socket
-a	Show all sockets
-r	Show routing table (netstat)
-i	Show interface statistics (netstat)

COMMAND COMPARISON

Command	Purpose	Best For
ss	Modern tool to show socket stats	All modern systems
ss -tuln	Show listening TCP/UDP ports	Check open ports
netstat	Traditional network statistics	Legacy systems, routing
lsof	List open files and connections	Find process using ports

Why use ss instead of netstat?

- Faster and more detailed output
- Part of iproute2 (more features)
- Works better with large number of connections

EXAMPLES IN ACTION

ss	Show all active connections	ss
ss -tuln	Show all listening TCP/UDP ports	ss -tuln
ss -plant	Show listening ports with process	ss -plant
netstat -tuln	Show listening TCP/UDP ports	netstat -tuln
netstat -r	Show routing table	netstat -r
lsof -i	Show all network connections	lsof -i
lsof -i :80	Show who is using port 80	lsof -i :80
ss -s	Show summary of socket usage	ss -s

UNDERSTANDING OUTPUT (ss -tuln)

Column	Meaning
Netid	Protocol (tcp / udp)
State	State of connection (LISTEN, ESTAB, etc.)
Recv-Q	Receive Queue (pending packets)
Send-Q	Send Queue (pending packets)
Local Address:Port	Local IP address and port
Peer Address:Port	Remote IP address and port

COMMON STATES (TCP)

State	Meaning
LISTEN	Waiting for incoming connections
ESTAB	Connection is established
SYN-SENT	Connection request sent
SYN-RECV	Connection request received
CLOSE-WAIT	Waiting for local close
TIME-WAIT	Waiting for connection to close

CHECK OPEN PORTS (QUICK GUIDE)

- Use ss -tuln to see open ports.
- Look for LISTEN state.
- 0.0.0.0 means open on all interfaces.
- Check process with ss -plant or lsof -i.

COMMON PORTS REFERENCE

Port	Service	Protocol	Description
22	SSH	TCP	Secure Shell
53	DNS	TCP/UDP	Domain Name System
80	HTTP	TCP	Web Traffic
443	HTTPS	TCP	Secure Web Traffic
3306	MySQL	TCP	MySQL Database
5432	PostgreSQL	TCP	PostgreSQL Database

TROUBLESHOOTING TIPS

- Port not listening? → Check service status.
- Connection refused? → Check firewall.
- Can't connect? → Check IP, port and service.
- Use ss -plant to find the process.
- Use netstat -r to check routing.

ONE-LINERS YOU'LL LOVE

ss -tuln	View all listening ports
ss -plant	View listening ports with process
lsof -i :22	Find who is using port 22
netstat -tuln	View all listening ports
ss -s	Show socket summary
watch -n 2 ss -tuln	Monitor ports every 2 sec

PRO TIP

Use ss -tuln for daily checks and lsof -i to find the process behind any port.

NETWORK CONNECTIONS WORKFLOW

1. Check Listening Ports (ss -tuln)
2. Find Process (ss -plant / lsof -i)
3. Check Firewall (ufw / iptables)
4. Test Connection (ping / telnet / nc)
5. Problem Solved! Connection OK

Know your ports, control your connections, secure your system!

Linux™ FIREWALL (UFW) COMMANDS

Manage Your Firewall Easily with UFW

Learn | Practice | Use | Explore | Master

1. ufw – CHECK STATUS

Check the current status of the firewall.

Syntax

```
ufw status
```

Example

```
$ ufw status
Status: active
To                         Action      From
22/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
```

Shows firewall status and rules.

2. ufw status verbose

Check firewall status in detailed (verbose) mode.

Syntax

```
ufw status verbose
```

Example

```
$ ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip
```

Shows detailed firewall information.

3. ufw allow

Allow connections on a specific port or service.

Syntax

```
ufw allow {port}/{proto}
```

Example

```
$ ufw allow 22/tcp
Rule added

$ ufw allow http
Rule added
```

Allows traffic on a port or service.

4. ufw deny

Deny (block) connections on a specific port or service.

Syntax

```
ufw deny {port}/{proto}
```

Example

```
$ ufw deny 23/tcp
Rule added

$ ufw deny telnet
Rule added
```

Blocks traffic on a port or service.

5. ufw delete

Delete a specific rule from the firewall.

Syntax

```
ufw delete {rule_number}
```

Example

```
$ ufw status numbered
[ 1] 22/tcp        ALLOW IN    Anywhere
[ 2] 80/tcp        ALLOW IN    Anywhere
[ 3] 23/tcp        DENY IN     Anywhere
$ ufw delete 3
Rule deleted
```

Deletes rule by number.

6. ufw reset

Reset firewall to default rules.

Syntax

```
ufw reset
```

Example

```
$ ufw reset
Resetting all rules to installed defaults. (Be sure to update your rules according to your needs.)
Firewall reloaded
```

Resets all rules to default.

COMMON RULE EXAMPLES

Command	Description
ufw allow 22/tcp	Allow SSH (port 22)
ufw allow 80/tcp	Allow HTTP (port 80)
ufw allow 443/tcp	Allow HTTPS (port 443)
ufw deny 23/tcp	Block Telnet (port 23)
ufw allow from 192.168.1.10	Allow from specific IP
ufw deny from 192.168.1.20	Block from specific IP
ufw allow 53/udp	Allow DNS (port 53 UDP)
ufw enable	Enable the firewall
ufw disable	Disable the firewall

FIREWALL STATES

State	Meaning
active	Firewall is enabled and rules are active
inactive	Firewall is disabled
enabled	Firewall will start on boot
disabled	Firewall will not start on boot

UFW RULE FORMAT

```
ufw [action] [from/to] [port] [proto] [options]
```

- action : allow | deny | reject | limit
- from/to : from <IP> | to <IP> (optional)
- port : Port number or service name
- proto : tcp | udp | any (default: any)
- options : log, comment, etc.

CHECK RULES WITH NUMBERS

```
$ ufw status numbered
Status: active
To                         Action      From
[ 1] 22/tcp                ALLOW IN    Anywhere
[ 2] 80/tcp                ALLOW IN    Anywhere
[ 3] 443/tcp               ALLOW IN    Anywhere
[ 4] 23/tcp                DENY IN     Anywhere
[ 5] 53/udp                ALLOW IN    Anywhere
```

Use numbers with 'ufw delete <number>' to remove rules.

PROFILES (APPLICATION BASED)

List all application profiles:

```
$ ufw app list
```

```
Available applications:
  Apache Full
  Apache Secure
  OpenSSH
```

LOGGING

Enable logging:

```
$ ufw logging on
```

Disable logging:

```
$ ufw logging off
```

DEFAULT POLICY

Set default incoming policy:

```
$ ufw default deny incoming
```

Set default outgoing policy:

```
$ ufw default allow outgoing
```

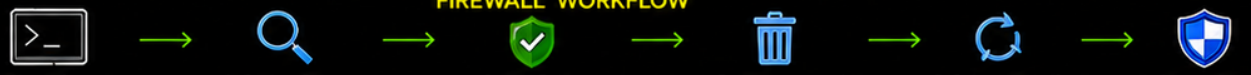
BEST PRACTICES

- Allow only necessary ports.
- Use specific IP rules for better security.
- Enable logging to monitor blocked attempts.
- Regularly review and clean up old rules.
- Keep your system and firewall updated.

PRO TIP

Enable UFW and allow only what you need. Less is more when it comes to security!

FIREWALL WORKFLOW



REMEMBER

A strong firewall today, a safer system tomorrow!

Linux™ ENVIRONMENT VARIABLES COMMANDS

Manage Your Environment Like a Pro

Learn | Practice | Use | Explore | Master

1. env – SHOW ALL VARIABLES

Display all environment variables in your current shell.

Syntax

```
env
```

Example

```
$ env | head -n 5
PATH=/usr/local/bin:/usr/bin:/bin
HOME=/home/user
USER=/user
SHELL=/bin/bash
LANG=en_US.UTF-8
```

Lists all environment variables.

2. printenv – PRINT VARIABLES

Print environment variables. Can show one or all variables.

Syntax

```
printenv [variable_name]
```

Example

```
$ printenv HOME
/home/user
$ printenv | head -n 5
PATH=/usr/local/bin:/usr/bin:/bin
HOME=/home/user
USER=/ser
SHELL=/bin/bash
```

Prints the value of a variable or all variables.

3. export – SET OR UPDATE VARIABLES

Create a new environment variable or update the value of existing one.

Syntax

```
export VAR_NAME="value"
```

Example

```
$ export MYVAR="Hello World"
$ printenv MYVAR
Hello World
$ export PATH="$PATH:/opt/mybin"
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/opt/mybin
```

Makes the variable available to child processes.

4. unset – REMOVE VARIABLES

Remove an environment variable.

Syntax

```
unset VAR_NAME
```

Example

```
$ export TEMPVAR="To be removed"
$ printenv TEMPVAR
To be removed
$ unset TEMPVAR
$ printenv TEMPVAR
(no output)
```

Deletes the variable from the environment.

5. COMMON VARIABLES

Variable	Description
$PATH	Directories to search for commands
$HOME	Home directory of the current user
$USER	Current logged in username
$SHELL	Current shell
$PWD	Present working directory
$LANG	Language and locale settings
$HOSTNAME	System hostname
$TERM	Terminal type
$EDITOR	Default text editor

Important variables used in daily operations.

EXAMPLES IN ACTION

Task	Commands
Show all variables	env
Show value of PATH	printenv PATH
Set a variable	export MYVAR="value"
Use a variable	echo $MYVAR
Add directory to PATH	export PATH="$PATH:/opt/bin"
Remove variable	unset MYVAR

COMMON VARIABLES EXPLAINED

Variable	Example Value	Explanation
$PATH	/usr/local/bin:/usr/bin:/bin	Where the system looks for commands
$HOME	/home/user	Your home directory
$USER	user	Current username
$SHELL	/bin/bash	Current shell in use
$PWD	/home/user/docs	Current working directory
$LANG	en_US.UTF-8	Language and locale
$HOSTNAME	linux-machine	System's hostname

PERSISTENT VS TEMPORARY VARIABLES

Temporary (Current Session)

Set with export
Available only in current terminal session.

```
$ export TEMPVAR="temp"
$ echo $TEMPVAR
temp
# Close terminal → variable gone
```

Persistent (All Sessions)

Add to shell profile file (e.g. ~/.bashrc, ~/.profile) to make it permanent.

```
$ echo 'export MYVAR="value"' >> ~/.bashrc
$ source ~/.bashrc
$ echo $MYVAR
value
# Available in every new session
```

WHERE VARIABLES ARE STORED?

System Wide

Stored in files like /etc/environment for all users.

User Specific

Stored in shell config files like ~/.bashrc or ~/.profile.

```
$ cat /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
LANG=en_US.UTF-8
```

User variables override system-wide variables.

SPECIAL VARIABLES

$$	Current process ID	echo $$
$?	Exit status of last command	echo $?
$#	Number of arguments passed	echo $#
$0	Name of the script/command	echo $0
$1, $2...	Arguments passed to script	echo $1
$*, $@	All arguments	echo $@

PRACTICAL EXAMPLE

```
$ echo "User: $USER"
User: user
$ echo "Home: $HOME"
Home: /home/user
$ echo "Shell: $SHELL"
Shell: /bin/bash
$ echo "Path: $PATH" | cut -d: -f1-3
Path: /usr/local/bin:/usr/bin:/bin
```

Shell → Reads Variables → Uses in Commands → Produces Output

Environment Variables
Store important system & user settings used by shell and applications.

BEST PRACTICES

- Always use meaningful variable names.
- Export only what is necessary.
- Keep PATH clean and add only required directories.
- Store permanent variables in ~/.bashrc or ~/.profile.
- Use unset to remove variables you no longer need.

BEST PRACTICES

PRO TIP
Use echo $VAR to quickly check the value of any variable.

ENVIRONMENT VARIABLE WORKFLOW

1. Check Variables (env / printenv)
2. Set or Update (export)
3. Use in Commands (echo $VAR)
4. Remove if Needed (unset)
5. Reload Shell (source ~/.bashrc)
6. Done!

REMEMBER
Environment variables control the behavior of your shell, programs and system.

Linux™ SHELL VARIABLES COMMANDS

Store, Use & Manage Data in Your Shell Like a Pro

Learn | Practice | Use | Explore | Master

1. variable=value – CREATE VARIABLE

Create a variable and assign a value to it.

Syntax

```
variable=value
```

Example

```
$ name=Lovejeet
$ age=32
$ city=Amritsar
$ echo $name
Lovejeet
```

No spaces around = sign.

2. echo $variable – DISPLAY VALUE

Display the value stored in a variable.

Syntax

```
echo $variable
```

Example

```
$ name=Linux
$ echo $name
Linux
$ echo $city
Amritsar
```

Use $ before the variable name.

3. read – TAKE INPUT FROM USER

Read input from user and store in a variable.

Syntax

```
read variable
```

Example

```
$ read username
Enter your name: Lovejeet
$ echo $username
Lovejeet
```

Useful for interactive scripts.

4. unset – REMOVE VARIABLE

Remove a variable from the shell.

Syntax

```
unset variable
```

Example

```
$ name=Hello
$ echo $name
Hello
$ unset name
$ echo $name
```

Variable and its value are removed.

5. export – MAKE ENV VARIABLE

Make a variable available to child processes.

Syntax

```
export variable
```

Example

```
$ export MYVAR=LinuxHub
$ env | grep MYVAR
MYVAR=LinuxHub
$ bash -c 'echo $MYVAR'
LinuxHub
```

Exported variables are environment variables.

VARIABLE BASICS

Variable Name Rules

- Start with a letter or underscore (_)
- Can contain letters, numbers, and _
- Case sensitive (name ≠ NAME)

Value Types

- String (text)
- Number
- Command output

Examples

```
name=Linux
count=10
path=/home/user
```

EXAMPLES IN ACTION

Task	Command	Output
Create variable	greet=Hello	—
Display value	echo $greet	Hello
Take input	read user echo "Hi $user"	(user input) Hi Lovejeet
Use in string	echo "Welcome $user"	Welcome Lovejeet
Remove variable	unset greet	—
Export variable	export LANG=en_US.UTF-8	—

VARIABLE TYPES

1. **Normal Variable**
   Available only in current shell.
   ```
   name=Linux
   ```
2. **Environment Variable**
   Available in current shell and child processes.
   ```
   export PATH=/usr/local/bin
   ```
3. **Readonly Variable**
   Cannot be changed or unset.
   ```
   readonly OS=Linux
   ```

USEFUL OPERATIONS

Operation	Example	Result
Get length	${#var}	Length of value
Default value	${var:-default}	Use default if empty
Alternate value	${var:+alt}	Use alt if set
Substring	${var:start:len}	Extract part
Replace	${var/old/new}	Replace once
Remove prefix	${var#prefix}	Remove smallest prefix
Remove suffix	${var%suffix}	Remove smallest suffix

COMMON BUILT-IN VARIABLES

Variable	Description
$0	Name of the script/shell
$1, $2...	Positional parameters
$#	Number of arguments
$@	All arguments
$$	Current process ID
$?	Exit status of last command
$USER	Current username
$HOME	Home directory
$PATH	Directories to search commands
$SHELL	Current shell

PRACTICAL EXAMPLE SCRIPT

```
#!/bin/bash
# Shell Variables Example
read -p "Enter your name: " name
read -p "Enter your city: " city
greet="Welcome $name"
echo $greet
echo "You are from $city"
export MY_CITY=$city
echo "MY_CITY exported: $MY_CITY"
```

OUTPUT

```
$ bash demo.sh
Enter your name: Lovejeet
Enter your city: Amritsar
Welcome Lovejeet
You are from Amritsar
MY_CITY exported: Amritsar
```

VARIABLE SCOPE

Normal variables are limited to current shell.

Exported variables are available to child processes.

BEST PRACTICES

- Use meaningful variable names.
- Quote variable expansions: "$var"
- Export only what is necessary.
- Unset variables you no longer need.
- Use readonly for important constants.

BEST PRACTICE

PRO TIP

Use double quotes "$var" to avoid word splitting and preserve spaces.

SHELL VARIABLES WORKFLOW

1. Create Variable (variable=value)
2. Use / Display (echo $variable)
3. Get Input (read variable)
4. Export (if needed) (export variable)
5. Remove (if not needed) (unset variable)
6. Done!

REMEMBER

Variables make your scripts powerful, flexible and reusable!

Linux™ COMMAND HISTORY

View, Search & Reuse Your Previous Commands Like a Pro

Learn | Practice | Use | Explore | Master

1. history – SHOW HISTORY

Display a list of previously executed commands.

Syntax

```
history [options]
```

Example

```
$ history
 1  ls -l
 2  cd /home/user
 3  nano file.txt
 4  gcc main.c -o main
 5  ./main
```

Shows numbered list of all commands.

2. !! – REPEAT LAST COMMAND

Repeat the last command executed.

Syntax

```
!!
```

Example

```
$ ls -l
(output)
$ !!
ls -l
(same output again)
```

Very quick way to run last command.

3. !number – RUN COMMAND BY NUMBER

Run a command from history by its number.

Syntax

```
!number
```

Example

```
$ history
101  git status
102  git add .
103  git commit -m "update"
104  git push
$ !102
git add .
```

Runs the command at that number.

4. Ctrl + R – SEARCH HISTORY

Search your history in reverse order (interactive search).

Syntax

```
Ctrl + R
```

Example

```
(Press Ctrl + R)
Type: git
(Found: git commit -m "update")
Press Enter to run
```

Great for searching long history.

5. !string – RUN LAST MATCH

Run the last command that starts with the string.

Syntax

```
!string
```

Example

```
$ !git
git push origin main
```

Runs the last command starting with string.

6. history -c – CLEAR HISTORY

Clear the current command history.

Syntax

```
history -c
```

Example

```
$ history -c
$ history
(no output)
```

Clears history from current session.

HISTORY OPTIONS

Option	Description
history	Show command history
history -n	Read new history lines from history file
history -r	Read history from history file
history -w	Write history to history file
history -c	Clear current history list
history -d offset	Delete entry at offset
history | less	View history page by page

EXAMPLES IN ACTION

```
$ history | tail -n 5
  150  docker ps
  151  docker images
  152  cd /var/www
  153  ls -la
  154  nano index.html
$ !!
nano index.html
$ !152
cd /var/www
$ Ctrl + R
(reverse search: docker)
```

HOW HISTORY WORKS

You run commands → Bash saves them in memory → Saved in ~/.bash_history (file)

~/.bash_history
Stores your permanent command history.

HISTORY FILE LOCATION

Default history file:

```
~/.bash_history
```

View the file:

```
cat ~/.bash_history
```

This file keeps your history between terminal sessions.

HISTORY SETTINGS (BASH)

Variable	Description
HISTSIZE	Number of commands stored in memory
HISTFILESIZE	Number of commands saved in ~/.bash_history
HISTCONTROL	Control behavior of history (ignorespace, ignoredups, etc.)

Example:

```
$ export HISTSIZE=1000
$ export HISTFILESIZE=2000
$ export HISTCONTROL=ignoredups:ignorespace
```

TIPS & BEST PRACTICES

- Use Ctrl + R to quickly find long commands.
- Use !! to save time repeating the last command.
- Increase HISTSIZE for more history in current session.
- Backup ~/.bash_history regularly if important.

COMMON SHORTCUTS SUMMARY

Shortcut	Purpose
history	Show all previous commands
!!	Repeat last command
!number	Run command by number
!string	Run last command starting with string
Ctrl + R	Search command history
history -c	Clear history

TERMINAL APPEARANCE

```
user@linux:~$ history | tail -n 3
  196  sudo systemctl restart nginx
  197  nano /etc/nginx/nginx.conf
  198  systemctl status nginx
user@linux:~$ !!
systemctl status nginx
● nginx.service - A high performance web server
   Active: active (running) since ...
```

DID YOU KNOW?

You can combine history with other commands!

```
$ history | grep git      # Find git commands
$ !?docker?               # Run last command containing "docker"
```

PRO TIP

Learn a few history shortcuts and you'll save hours in the terminal! ⚡

COMMAND HISTORY WORKFLOW

1. Run Commands (Work as usual)
2. View History (history)
3. Search or Find (Ctrl + R / !string)
4. Run Again (!! / !number)
5. Saved to File (~/.bash_history)
6. Ready for Next Session!

REMEMBER

Every command you run today can save you time tomorrow! 💪

SHELL BASICS

Learn the Foundations of Linux Shell Like a Pro

Learn | Practice | Use | Explore | Master

1. WHAT IS SHELL?

Shell is a command line interpreter that acts as an interface between user and the Linux kernel.

You type commands in the shell, shell executes them for the system.

2. SHELL vs TERMINAL

SHELL	TERMINAL
• Command interpreter	• Application (emulator)
• Executes commands	• Provides UI to interact with shell
• Examples: bash, sh, zsh	• Examples: GNOME Terminal, Konsole, xTerm
• Runs inside terminal	

Terminal is the window, Shell is the program inside it.

3. COMMON SHELLS

- **Bash** — Bourne Again SHell (Default in most Linux distros)
- **sh** — Bourne Shell (Original Unix shell)
- **zsh** — Z Shell (Advanced features & customization)
- **fish** — Friendly Interactive Shell (User-friendly & smart autosuggestions)

Check your current shell: `echo $SHELL`

4. THE SHELL PROMPT

Prompt shows that shell is ready to accept your command.

```
user@linux:~$    # Bash prompt (default)
user@linux:~#    # Root prompt
user@linux:~%    # Zsh prompt
user@linux:~>    # Custom prompt example)
```

Prompt Meaning:
- `$` -> Regular user
- `#` -> Root (superuser)
- `%` or `>` -> Depends on shell/theme

5. CLI (COMMAND LINE INTERFACE)

CLI lets you control your system by typing commands instead of using GUI.

CLI is powerful, fast and works on servers without GUI.

6. BASIC COMMAND STRUCTURE

Every command follows a simple structure:

```
command [options] [arguments]
```

What to do | How to do it | On what

Example:

```
$ ls -l /home/user
```

command | option | argument

7. RUN YOUR FIRST COMMANDS

Command	Purpose
echo Hello, Linux!	Print text to the screen
pwd	Show current directory
ls	List files and folders
whoami	Show current user
date	Show current date & time
clear	Clear the terminal screen

8. COMMAND HELP

Get help for any command:

```
$ command --help    # Help for a command
$ man command       # Detailed manual (man page)
$ info command      # Info documentation
```

Example:
`$ man ls`

9. REDIRECTION & PIPES (BASICS)

Redirection (>, >>, <)

```
$ echo "Hello" > file.txt     # Write to file
$ echo "World" >> file.txt    # Append to file
$ cat < file.txt              # Read from file
```

Pipes (|)

```
$ ls -l | less                # Send output to another command
$ cat file.txt | grep "Hello"
```

10. SHELL FEATURES

- **Command History** — Access previous commands
- **Tab Completion** — Auto-complete files & commands
- **Aliases** — Create shortcuts for commands
- **Scripting** — Automate tasks with shell scripts

BEST PRACTICES

- Use meaningful names for scripts & variables.
- Use Tab completion to save time.
- Read man pages to learn more.
- Practice regularly to get comfortable with CLI.

BEST PRACTICES

POPULAR TERMINALS

GNOME Terminal | Konsole | xTerm | Alacritty | Terminator

CHECK YOUR SHELL

```
$ echo $SHELL
/bin/bash
```

Change your shell (if installed):

```
$ chsh -s /bin/zsh
```

QUICK TIPS

- Use ↑ / ↓ arrow keys to navigate history.
- Press Ctrl + C to cancel a command.
- Press Ctrl + L to clear the screen.
- Type `exit` or `logout` to close the shell.

SHELL BASICS WORKFLOW

Open Terminal → See the Prompt → Type a Command → Shell Executes the Command → Output is Displayed → Learn & Explore More → Become a Command Line Pro!

PRO TIP
The more you use the shell, the more powerful and faster you become!

REMEMBER
Master the shell, master Linux!

Linux BASH BOURNE AGAIN SHELL

Powerful Scripting & Automation

Terminal | Script | Run | Functions | Variables | Automation

1. WHAT IS BASH?

Bash (Bourne Again Shell) is the default command interpreter in most Linux distributions.
It lets you run commands, automate tasks and write shell scripts.

Bash is both a command language and a scripting language.

2. SHELL SCRIPT HEADER

Every Bash script starts with shebang.

```
#!/bin/bash
# This tells the system to use
# Bash to run the script.
```

Always make your script executable before running.

3. RUN A BASH SCRIPT

Make script executable and run it.

Syntax

```
chmod +x script.sh
./script.sh
```

Or run with bash:

```
bash script.sh
```

Use bash script.sh when the script is not executable.

4. VARIABLES

Store data and use it in scripts.

Syntax

```
name="Lovejeet"
age=32
echo "Name: $name"
echo "Age: $age"
```

No spaces around = sign.
Use $ to access variable.

5. COMMON COMMANDS

Command	Description
echo	Display text
read	Read input from user
ls	List files
cd	Change directory
pwd	Print working directory
cp	Copy files or directories
mv	Move or rename files
rm	Remove files
mkdir	Create directory

6. EXAMPLE SCRIPT

Simple Bash script example.

```
#!/bin/bash
# This is a simple script
name="Linux User"
echo "Hello, $name!"
echo "Today is $(date)"
echo "Your current directory is:"
pwd
```

Save as hello.sh
Make executable: chmod +x hello.sh
Run with: ./hello.sh

7. USER INPUT (read)

Take input from user.

Syntax

```
read variable_name
echo "You entered: $variable_name"
```

Example

```
read name
echo "Hello, $name!"
```

Use -p for prompt message.

8. IF-ELSE STATEMENT

Make decisions in your scripts.

Syntax

```
if [ condition ]; then
    # commands
else
    # commands
fi
```

Example

```
if [ $age -ge 18 ]; then
    echo "Adult"
else
    echo "Minor"
fi
```

9. FOR LOOP

Repeat commands for items.

Syntax

```
for item in list; do
    # commands using $item
done
```

Example

```
for i in 1 2 3 4 5; do
    echo "Number: $i"
done
```

10. FUNCTIONS

Organize code into reusable blocks.

Syntax

```
my_function() {
    echo "This is a function"
}
my_function
```

Example

```
greet() {
    echo "Hello from function!"
}
greet
```

11. COMMENTS

Add notes in your script.

Syntax

```
# This is a single line comment

: <<'COMMENT'
This is a
multi-line comment
COMMENT
```

Comments are ignored when the script runs.

12. EXIT STATUS

Every command returns a status.

Syntax

```
command
echo $?
```

Example

```
ls /notfound
echo $?
```

0 = Success
Non-zero = Error

VARIABLES

Variable	Description
$0	Name of the script
$1, $2, ...	Positional parameters
$#	Number of arguments
$@	All arguments
$$	Process ID of current script
$?	Exit status of last command
$USER	Current logged in user
$PWD	Present working directory
$HOME	Home directory

SCRIPT ARGUMENTS

Pass values to your script.

Example (args.sh)

```
#!/bin/bash
echo "Script name: $0"
echo "First arg  : $1"
echo "Second arg: $2"
echo "Total args: $#"
```

Run: bash args.sh one two
Output:
Script name: args.sh
First arg : one
Second arg : two
Total args:: 2

USEFUL SHORTCUTS

Shortcut	Description
Ctrl + C	Cancel current command
Ctrl + D	Logout / Exit shell
Ctrl + L	Clear screen
Ctrl + R	Search command history
Tab	Auto-complete files/commands
↑ / ↓	Previous / Next command
!!	Repeat last command
!n	Repeat command number n

BEST PRACTICES

- Use meaningful variable and function names.
- Add comments to explain your code.
- Use quotes to handle spaces in variables.
- Check exit status to handle errors.
- Keep scripts small and focused.

BEST PRACTICES

BASH IS USED FOR

- System administration
- Task automation
- Log processing
- File handling
- DevOps & Deployment

PRO TIP

Practice small scripts daily. Automation saves time and reduces mistakes!

BASH WORKFLOW

1. Create Script (.sh file)
2. Write Code (Commands)
3. Save Script
4. Make Executable (chmod +x script.sh)
5. Run Script (./script.sh or bash script.sh)
6. Done!

REMEMBER

Test your scripts before using in production. Backup important files and keep learning Bash!

Linux SHELL SCRIPTS

Automate Tasks • Save Time • Boost Productivity

Terminal | Script | Automation | Executable | Powerful

1. WHAT IS A SHELL SCRIPT?

A shell script is a text file that contains commands for the shell to execute.
It helps you automate tasks, save time, and reduce manual work.

Bash (Bourne Again Shell) is the most popular shell for scripting.

2. SHEBANG (SCRIPT HEADER)

Every shell script should start with shebang. It tells the system which interpreter to use.

Syntax

```
#!/bin/bash
```

This line must be the first line in your script.

3. CREATE A SCRIPT

Example: script.sh

```
#!/bin/bash
# My First Script
echo "Hello, World!"
echo "Today is $(date)"
echo "Current user: $(whoami)"
```

echo is used to display text or output in the terminal.

4. MAKE SCRIPT EXECUTABLE

Give execute permission to the script.

Syntax

```
chmod +x script.sh
```

Example

```
$ ls -l script.sh
-rw-r--r-- 1 user user 202 May 19 10:30 script.sh
$ chmod +x script.sh
$ ls -l script.sh
-rwxr-xr-x 1 user user 202 May 19 10:31 script.sh
```

chmod +x makes the script executable.

5. RUN THE SCRIPT

Run your script using one of these ways.

Syntax

```
./script.sh      # Run from current directory
bash script.sh   # Run using bash
sh script.sh     # Run using sh
```

Example

```
$ ./script.sh
Hello, World!
Today is Sun May 19 10:31:45 AM IST 2024
Current user: user
```

./script.sh runs the script in the current directory.

6. VARIABLES IN SHELL SCRIPT

Store and use data in your scripts.

Example

```
#!/bin/bash
name="Lovejeet"
age=32
echo "Name: $name"
echo "Age: $age"
echo "Home: $HOME"
```

Use $variable to access the value.

7. COMMON COMMANDS

Command	Description
echo	Display text or output
read	Take input from user
ls	List files and directories
cd	Change directory
pwd	Print working directory
cp	Copy files or directories
mv	Move or rename files
rm	Remove files
mkdir	Create directory

8. CONTROL STRUCTURES

IF-ELSE

```
if [ $age -ge 18 ]; then
  echo "Adult"
else
  echo "Minor"
fi
```

WHILE LOOP

```
count=1
while [ $count -le 5 ]; do
  echo "Count: $count"
  count=$((count+1))
done
```

FOR LOOP

```
for item in 1 2 3 4 5; do
  echo "Item: $item"
done
```

CASE

```
case $1 in
  start) echo "Start" ;;
  stop) echo "Stop" ;;
  *) echo "Invalid" ;;
esac
```

9. COMMENTS

Comments are ignored by the shell.

Syntax

```
# This is a comment
```

Example

```
#!/bin/bash
# This script prints a message
# Written by Lovejeet
echo "Hello, Script!"
```

Use comments to explain your code.

10. PRACTICAL EXAMPLE

Backup script example

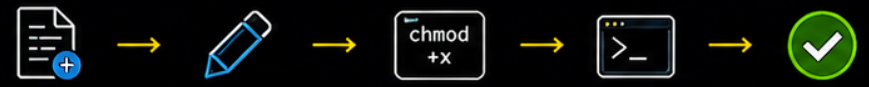
```
#!/bin/bash
# Backup script
source_dir="$HOME/Documents"
dest_dir="$HOME/Backups"
mkdir -p "$dest_dir"
cp -r "$source_dir" "$dest_dir/"
echo "Backup completed!"
```

This script copies Documents folder to Backups folder.

BENEFITS OF SHELL SCRIPTS

- Automate repetitive tasks
- Save time and improve productivity
- Easy to write and understand
- Powerful and flexible
- Works on most Linux distributions

SHELL SCRIPT WORKFLOW



1. Create Script (.sh file)
2. Write Commands (Use echo, ls, etc.)
3. Make Executable (chmod +x script.sh)
4. Run Script (./script.sh)
5. Automation Done!

TIPS

- Always test your script.
- Use meaningful names.
- Add comments.
- Check permissions.
- Keep it simple!

REMEMBER

```
#!/bin/bash
chmod +x script.sh
#/script.sh
```

★ Practice makes perfect!

Shell Scripts = Power of Commands + Automation + Productivity

Linux SHELL VARIABLES

Store Data • Reuse Values • Make Scripts Smarter

Terminal | Script | Variables | Automation | Productive

1. WHAT IS A SHELL VARIABLE?

A shell variable is a name that stores a value. You can use the value later in your commands, scripts and terminal.

name → Linux

(Variable Name) (Value)

Variables help store and reuse values easily. They make scripts dynamic.

2. VARIABLE ASSIGNMENT

Assign a value to a variable using = (no spaces around).

Syntax

```
variable_name="value"
```

Example

```
name="Linux"
version="5.15"
msg="Hello World"
```

No spaces around = sign.

3. ACCESSING VARIABLES

Use $ before the variable name to get its value.

Syntax

```
echo $variable_name
```

Example

```
name="Linux"
echo $name        # Output: Linux
```

$ is used to expand the variable and get its value.

4. COMMON EXAMPLES

Example	Output
`name="Linux"` `echo $name`	Linux
`greet="Hello, $name!"` `echo $greet`	Hello, Linux!

`a=10`
`b=20`
`echo $((a + b))`
	30

You can use variables in text, commands and expressions.

5. READ INPUT INTO VARIABLES

Take input from user and store in variable.

Syntax

```
read variable_name
```

Example

```
echo -n "Enter your name: "
read username
echo "Hello $username"
```

read command stores user input into the variable.

6. USEFUL PREDEFINED VARIABLES

Variable	Description
$USER	Current logged in username
$HOME	Home directory of current user
$PWD	Present working directory
$PATH	Directories in which shell looks for commands
$SHELL	Current shell
$?	Exit status of last command
$$	Process ID of current shell

Predefined variables are set by the system.

7. EXAMPLES OF PREDEFINED VARIABLES

Example	Output (Example)
echo $USER	lovejeet
echo $HOME	/home/lovejeet
echo $PWD	/home/lovejeet/projects
echo $PATH	/usr/local/bin:/usr/bin:/bin:...
echo $SHELL	/bin/bash
echo $$	12345
echo $?	0

Values may vary on your system.

8. EXPORT VARIABLES (ENVIRONMENT)

Use export to make a variable available to child processes and scripts.

Example

```
export MY_VAR="Hello"
echo $MY_VAR          # Hello
```

Exported variables are called environment variables.

9. UNSET VARIABLES

Remove a variable using unset.

Example

```
name="Linux"
unset name
echo $name      # (no output)
```

Unset removes the variable from current shell.

10. VARIABLE RULES

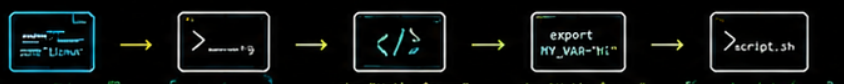
- Variable names can contain letters, numbers and underscores.
- Must start with a letter or underscore.
- Case sensitive (Name and name are different).
- No spaces around = sign.
- Avoid using special characters.

Follow these rules to avoid errors.

QUICK REFERENCE

```
name="Linux"          # Assignment
echo $name            # Access value
read var              # Read input
export VAR="val"      # Export variable
unset var             # Remove variable
```

SHELL VARIABLE WORKFLOW



1. Assign (Value store)
2. Use (Access value)
3. Use in Script (Dynamic output)
4. Export (For other processes)
5. Run Script (Use variable)

TIPS

- Use meaningful names.
- Use quotes for text values.
- Use $ to access values.
- Export if needed in scripts.
- Keep scripts clean and simple.

EXAMPLE SCRIPT (demo.sh)

```
#!/bin/bash
name="Linux"
version="5.15"
echo "Name: $name"
echo "Version: $version"
echo "User: $USER"
echo "Home: $HOME"
echo "Path: $PATH"
```

```
$ chmod +x demo.sh
$ ./demo.sh
```

PRO TIP

Variables make your scripts flexible and reduce repeated work.

> "Variables are like memory in your shell – store once, use anywhere!"

NOTE

Variable values exist only in the current shell session unless exported.

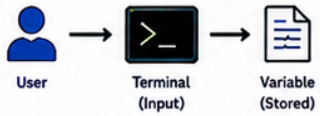
Linux USER INPUT

Take Input from User in Shell Scripts

Terminal | Input | Script | User | Data | Interactive Script

1. WHAT IS USER INPUT?

User input allows a script to take data from the user during execution.
It makes scripts interactive and flexible.



It helps scripts to get data like name, age, choice, file, etc.

2. BASIC read COMMAND

The read command reads a line from user input and stores it in a variable.

Syntax

```
read variable_name
```

Example

```
$ read name
echo "Hello $name"
```

```
$ ./script.sh
Enter your name: Lovejeet
Hello Lovejeet
```

Press Enter to submit the input.

3. read -p (WITH PROMPT)

Use -p option to show a prompt message to the user.

Syntax

```
read -p "Prompt message: " variable
```

Example

```
$ read -p "Enter your age: " age
echo "Your age is $age"
```

```
$ ./script.sh
Enter your age: 25
Your age is 25
```

-p displays a prompt and waits for input.

4. MULTIPLE INPUTS

You can read multiple values in one line.

Syntax

```
read var1 var2 var3
```

Example

```
$ read -p "Enter name, age, city: " name age city
echo "Name: $name"
echo "Age: $age"
echo "City: $city"
```

```
$ ./script.sh
Enter name, age, city: Lovejeet 25 Amritsar
Name: Lovejeet
Age: 25
City: Amritsar
```

Separate values with space.

5. HIDDEN INPUT (PASSWORD)

Use -s option to hide user input (useful for passwords).

Syntax

```
read -s -p "Password: " pass
```

Example

```
$ read -s -p "Enter password: " pass
echo
echo "Password saved!"
```

```
$ ./script.sh
Enter password: ********
Password saved!
```

Input is hidden. Nothing is shown on the screen.

6. DEFAULT VALUE

Provide a default value if user enters nothing.

Example

```
read -p "Enter username: " user
echo $(user:-guest)
```

```
$ ./script.sh
Enter username:
guest
```

If input is empty, 'guest' is used.

7. LIMIT INPUT (FIRST WORD ONLY)

By default, read takes the whole line.
Use IFS to limit input.

Example

```
IFS= read -r name
echo "Hello $name"
```

```
$ ./script.sh
Enter your name: Lovejeet Singh
Hello Lovejeet
```

Reads only the first word.

8. VALIDATE USER INPUT

Check and validate input using conditions.

Example

```
read -p "Enter age: " age
if [[ $age -lt 18 ]]; then
   echo "You are not allowed."
else
   echo "Access granted."
fi
```

```
$ ./script.sh
Enter age: 16
You are not allowed.
```

Always validate important input.

9. USE IN LOOPS

User input can be used inside loops.

Example

```
while true; do
  read -p "Enter a number (or 'q' to quit): " n
  [ "$n" = "q" ] && break
  echo "You entered: $n"
done
```

```
$ ./script.sh
Enter a number (or 'q' to quit): 10
You entered: 10
Enter a number (or 'q' to quit): q
```

Loop runs until user types 'q'.

10. PRACTICAL EXAMPLE

Simple interactive script example.

```
#!/bin/bash
read -p "Enter your name: " name
read -p "Enter your city: " city
echo "------------------------"
echo "Hello $name!"
echo "You are from $city."
echo "Welcome!"
```

```
$ ./script.sh
Enter your name: Lovejeet
Enter your city: Amritsar
------------------------
Hello Lovejeet!
You are from Amritsar.
Welcome!
```

Collects and shows user information.

COMMON read OPTIONS

-p	Show prompt message
-s	Silent input (hidden)
-r	Do not treat backslash as escape
-n	Read N characters only
-t	Timeout in seconds

USER INPUT WORKFLOW

BEST PRACTICES

- Always prompt clearly.
- Validate user input.
- Use quotes with variables.
- Handle empty input.
- Keep scripts user-friendly.

PRO TIP

Use read -p for better user experience and always validate important input to avoid errors.

Good Input → Better Scripts → Happy Users

Linux Shell Scripting

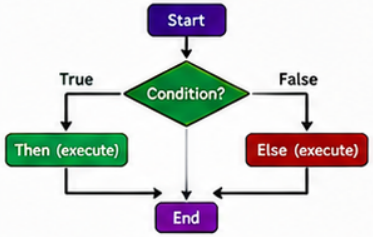
if / else CONDITIONS

Make Decisions in Your Scripts

Terminal | Script | Logic | Condition | True / False

1. WHAT IS IF / ELSE?

if / else allows your script to make decisions.
It checks a condition.
If True → then block runs
If False → else block runs



It is the most basic and powerful decision-making tool.

2. BASIC IF - ELSE SYNTAX

```
if [ condition ]; then
   # commands if condition is True
else
   # commands if condition is False
fi
```

Keyword	Meaning
if	Start of condition
then	If condition is True, run this block
else	If condition is False, run this block
fi	End of if block

3. EXAMPLE: NUMBER CHECK

Check if a number is positive or not.

```
#!/bin/bash
read -p "Enter a number: " num

if [ $num -gt 0 ]; then
   echo "Positive Number"
else
   echo "Not a Positive Number"
fi
```

Output (Example)

```
Enter a number: 10
Positive Number
```

4. EXAMPLE: STRING CHECK

Check if a string is equal to "Linux".

```
#!/bin/bash
read -p "Enter your OS:: " os

if [ "$os" == "Linux" ]; then
   echo "Great! You are using Linux"
else
   echo "You are Pusing something else"
fi
```

Output (Example)

```
Enter your OS: Linux
Great! You are using Linux
```

5. IF - ELIF - ELSE LADDER

Check multiple conditions.

```
#!/bin/bash
read -p "Enter marks (0-100): " m

if [ $m -ge 90 ]; then
   echo "Grade: A"
elif [ $m -ge 75 ]; then
   echo "Grade: B"
elif [ $m -ge 50 ]; then
   echo "Grade: C"
else
   echo "Grade: F"
fi
```

Output (Example)

```
Enter marks (0-100): 82
Grade: B
```

6. COMMON COMPARISON OPERATORS

Operator	Meaning	Example
-eq	Equal	[$a -eq $b]
-ne	Not equal	[$a -ne $b]
-gt	Greater than	[$a -gt $b]
-lt	Less than	[$a -lt $b]
-ge	Greater or equal	[$a -ge $b]
-le	Less or equal	[$a -le $b]

Use -eq for numbers and == for strings.

7. FILE CHECK CONDITIONS

Operator	Meaning	Example
-e	File exists	[-e file.txt]
-f	Is a regular file	[-f file.txt]
-d	Is a directory	[-d /home/user]
-r	Readable	[-r file.txt]
-w	Writable	[-w file.txt]
-x	Executable	[-x script.sh]

Useful for checking files and permissions.

8. EXAMPLE: FILE EXISTS CHECK

```
#!/bin/bash
read -p "Enter file name: " file

if [ -f "$file" ]; then
   echo "File '$file' exists."
else
   echo "File '$file' does not exist."
fi
```

Output (Example)

```
Enter file name: test.txt
File 'test.txt' exists.
```

9. NESTED IF EXAMPLE

```
#!/bin/bash
read -p "Enter age: " age
if [ $age -ge 18 ]; then
   echo "You are adult"
   if [ $age -ge 60 ]; then
      echo "You are a senior citizen"
   fi
else
   echo "You are minor"
fi
```

Output (Example)

```
Enter age: 65
You are adult
You are a senior citizen
```

10. BEST PRACTICES

- Always put spaces inside [ ].
  Correct: [$a -eq $b] Wrong: [$a -eq $b]
- Quote your variables.
- Use meaningful variable names.
- Indent your code for readability.
- Test all possible conditions.

GOOD LOGIC BETTER SCRIPTS

QUICK KEYWORDS

if	Start condition
then	If True, run this block
else	If False, run this block
elif	Else if (another condition)
fi	End of if block

IF / ELSE WORKFLOW

Start Your Script → Check Condition → True Run 'then' Block / False Run 'else' Block → End Script

TIPS

- Conditions return True (0) or False (non-zero).
- Use test command or [ ] for conditions.
- Debug with echo to check values.

REMEMBER

Every decision in a script starts with if.
Think before you code, that's the power of logic! 💪

Linux Shell Scripting

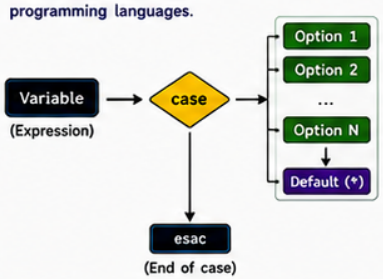
case Statement

Handle Multiple Conditions Easily

Terminal | Script | Conditions | Menu | Options

1. WHAT IS CASE STATEMENT?

The case statement is used to compare a variable with multiple values (patterns).
It is like switch-case in other programming languages.



It makes scripts clean and easy to read when you have many options.

2. SYNTAX

```
case expression in
   pattern1)
     commands
     ;;
   pattern2)
     commands
     ;;
     ...
   *)
     default commands
     ;;
esac
```

Keyword	Meaning
case	Start case statement
in	Begins the list of patterns
pattern)	A match pattern (option)
;;	End of commands for an option
*	Default option (if no match)
esac	End of case statement

3. EXAMPLE: SIMPLE MENU

Create a simple menu-driven script.

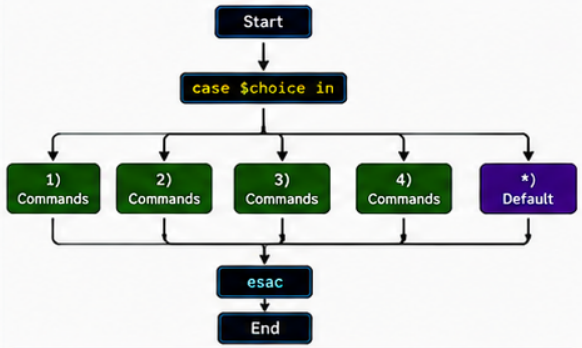
```
#!/bin/bash
echo "===== Main Menu ====="
echo "1) View System Info"
echo "2) Check Disk Space"
echo "3) Current Date & Time"
echo "4) Exit"
echo -n "Enter your choice [1-4]: "
read choice

case $choice  in
  1) uname -a ;;
  2) df -h ;;
  3) date ;;
  4) echo "Goodbye!" ;;
  *) echo "Invalid choice!" ;;
esac
```

Example Output

```
Enter your choice [1-4]: 2
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   20G   28G  42% /
....             ...                 /
```

4. HOW IT WORKS



- The expression is checked against each pattern.
- If match found, that block runs.
- If no match, * (default) block runs.

5. EXAMPLE: DAY CHECKER

Check day type using case statement.

```
#!/bin/bash
echo -n "Enter day (Mon..Sun): "
read day

case $day in
   Mon|Tue|Wed|Thu|Fri)
      echo "It's a Weekday" ;;
   Sat|Sun)
      echo "It's a Weekend" ;;
   *)
      echo "Invalid day!" ;;
esac
```

Example Output

```
Enter day (Mon..Sun): Sat
It's a Weekend
```

6. PATTERN EXAMPLES

Pattern	Meaning
word)	Matches exact word
a|b|c)	Matches any of a, b or c
*)	Default case (any value)
abc*)	Matches anything starting with abc
*xyz)	Matches anything ending with xyz
text)	Matches anything containing text

Patterns are case-sensitive.
Use lowercase/uppercase carefully.

7. EXAMPLE: FILE TYPE CHECK

Check type of a file.

```
#!/bin/bash
echo -n "Enter file name: "
read file

case $file in
  *.sh)
     echo "It's a Shell Script" ;;
  *.txt)
     echo "It's a Text File" ;;
  *.jpg|*.png)
     echo "It's an Image File" ;;
  *)
     echo "Unknown File Type" ;;
esac
```

Example Output

```
Enter file name: notes.txt
It's a Text File
```

8. NESTED CASE (ADVANCED)

Chase inside another case.

```
#!/bin/bash
echo -n "Select OS (1.Linux 2.Windows): "
read os

case $os in
  1) echo -n "Linux Distro? (1.Ubuntu 2.CentOS): "
     read dist
     case $dist in
       1) echo "Ubuntu Selected" ;;
       2) echo "CentOS Selected" ;;
       *) echo "Invalid Distro" ;;
     esac
     ;;
  2) echo "Windows Selected" ;;
  *) echo "Invalid Option" ;;
esac
```

Nested case helps in multi-level menus and more complex decisions.

9. BEST PRACTICES

- Use case when you have many options.
- Always close each block with ;;
- Always include a * (default) case.
- Keep menu simple and user-friendly.
- Indent properly for better readability.
- Test all possible options.

PRO TIP

case makes your script fast, clean and easy to maintain!

KEYWORDS QUICK REFERENCE

```
case      -> Start case statement
in        -> Begin patterns
pattern)  -> Option / match pattern
;;        -> End of option block
*         -> Default option
esac      -> End case statement
```

CASE STATEMENT WORKFLOW

Get Input (Expression) → Match with Patterns → Execute Matching Block → Finish & Exit esac

REMEMBER

case is your best friend when you need to compare one variable against many options!

Linux Shell Scripting

for LOOP

Repeat Tasks Easily with for Loop

Terminal | Script | Loop | Files | Numbers | Automation

1. WHAT IS FOR LOOP?

for loop is used to repeat a block of commands multiple times.
It works with a list of items (files, numbers, words, etc.).

for loop = Iteration + Automation

2. BASIC SYNTAX

Syntax

```
for variable in item1 item2 item3 ...
do
    commands
done
```

Keyword	Meaning
for	Start of loop
variable	Loop variable
in	List of items to iterate
do	Start of loop body
done	End of loop

Loop runs once for each item in the list.

3. EXAMPLE: LOOP THROUGH NUMBERS

Print numbers from 1 to 5.

Script

```
#!/bin/bash
for i in 1 2 3 4 5
do
   echo "Number: $i"
done
```

Output

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

4. EXAMPLE: LOOP THROUGH WORDS

Print each word.

Script

```
#!/bin/bash
for fruit in Apple Banana Mango
do
   echo "Fruit: $fruit"
done
```

Output

```
Fruit: Apple
Fruit: Banana
Fruit: Mango
```

5. EXAMPLE: LOOP THROUGH FILES

List all .sh files in the current directory.

Script

```
#!/bin/bash
for file in *.sh
do
   echo "File: $file"
done
```

Output (example)

```
File: backup.sh
File: install.sh
File: update.sh
```

6. EXAMPLE: LOOP THROUGH FILES (with ls)

Show file details.

Script

```
#!/bin/bash
for file in *
do
   echo "---- $file ----"
   ls -l "$file"
done
```

Useful for processing multiple files.

7. EXAMPLE: C-SYLE FOR LOOP

Loop using (( )) like in C language.

Script

```
#!/bin/bash
for (( i=1; i<=5; i++ ))
do
   echo "Count: $i"
done
```

Output

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

8. EXAMPLE: STEP VALUES

Count by step of 2.

Script

```
#!/bin/bash
for i in {0..10..2}
do
   echo "Step: $i"
done
```

Output

```
Step: 0
Step: 2
Step: 4
Step: 5
Step: 4
Step: 6
Step: 8
Step: 10
```

9. EXAMPLE: BACKUP MULTIPLE FILES

Backup all .txt files.

Script

```
#!/bin/bash
for file in *.txt
do
   cp "$file" "backup_$file"
   echo "Backed up: $file"
done
```

Output (example)

```
Backed up: notes.txt
Backed up: file1.txt
Backed up: file2.txt
```

10. NESTED FOR LOOP

Multiplication table (1 to 3).

Script

```
#!/bin/bash
for i in 1 2 3
do
   for j in 1 2 3
   do
      echo "$i x $j = $((i*j))"
   done
done
```

Output

```
1 x 1 = 1   1 x 2 = 2   1 x 3 = 3
2 x 1 = 2   2 x 2 = 4   2 x 3 = 6
3 x 1 = 3   3 x 2 = 6   3 x 3 = 9
```

COMMON USES

- Process list of files
- Backup multiple files
- Batch rename files
- Automation of tasks
- Data processing

FOR LOOP WORKFLOW

1. for starts
2. Get items from list
3. Run commands for each item
4. Next item
5. Loop ends (done)

BEST PRACTICES

- Always use quotes "$var"
- Use meaningful variable names
- Test with small data first
- Indent code for readability
- Handle files carefully

PRO TIP

for loop saves time and reduces manual work.
Use it for automation and smart scripting!

KEYWORDS QUICK REFERENCE

for	→	Start loop
in	→	List of items
do	→	Start block
done	→	End loop

Loop Smart, Work Less, Automate More!

Linux Shell Scripting

while & until

Repeat as Long as Needed!

Terminal | Script | Loop | Condition | Repeat | Automation

1. WHAT ARE WHILE & UNTIL LOOPB?

Loops are used to repeat a block of commands while a condition is True (while loop) or until a condition becomes True (until loop).

while	until
Runs while condition is True	Runs until condition becomes True

Use loops to automate repetitive tasks easily in shell scripts.

2. WHILE LOOP SYNTAX

```
while [ condition ]
  do
      commands
done
```

Keyword	Meaning
while	Start while loop
do	Start of loop body
condition	Test condition
done	End of loop

Loop continues while condition is True.

3. UNTIL LOOP SYNTAX

```
until [ condition ]
  do
      commands
done
```

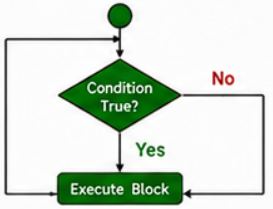
Keyword	Meaning
until	Start until loop
do	Start of loop body
condition	Test condition
done	End of loop

Loop continues until condition is True.

4. KEY DIFFERENCE

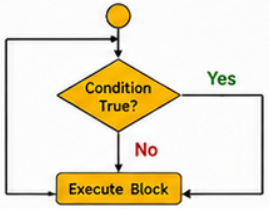
while Loop

Executes while the condition is True.



until Loop

Executes until the condition is True.



Think: while = repeat while true, until = repeat until true.

5. EXAMPLES: WHILE LOOP

A) Count from 1 to 5

```
i=1
while [ $i -le 5 ]
do
   echo "Number: $i"
   i=$((i+1))
done
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

B) Read user input until 'exit'

```
while true
do
   read -p "Enter something (type exit): " val
   [ "$val" = "exit" ] && break
   echo "You entered: $val"
done
```

```
Enter something (type exit): hello
You entered: hello
Enter something (type exit): Linux
You entered: Linux
Enter something (type exit): exit
```

C) Process files until none left

```
while ls *.txt 1> /dev/null 2>&1
do
   file=$(ls *.txt | head -n1)
   echo "Processing: $file"
   rm "$file"
done
echo "All files processed!"
```

```
Processing: file1.txt
Processing: file2.txt
Processing: file3.txt
All files processed!
```

6. EXAMPLES: UNTIL LOOP

A) Count until i becomes 5

```
i=1
until [ $i -gt 5 ]
do
   echo "Number: $i"
   i=$((i+1))
done
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

B) Wait until file exists

```
file="ready.txt"
until [ -f "$file" ]
do
   echo "Waiting for $file..."
   sleep 2
done
echo "$file is found!"
```

```
Waiting for ready.txt...
Waiting for ready.txt...
Waiting for ready.txt...
ready.txt is found!
```

C) Keep trying until success

```
count=1
until ping -c1 google.com &>/dev/null
do
   echo "Try $count: No connection"
   count=$((count+1))
   sleep 2
done
echo "Connected to internet!"
```

```
Try 1: No connection
Try 2: No connection
Try 3: No connection
Connected to internet!
```

7. LOOP STRUCTURE SUMMARY

```
while [ condition ]
do
    commands
done

# Runs while condition is True
```

```
until [ condition ]
do
    commands
done

# Runs until condition is True
```

8. COMMON USE CASES

- Repeat tasks multiple times
- Process lists and files
- Keep asking for user input
- Wait for conditions or events
- Automate monitoring tasks

9. LOOP WORKFLOW

while = repeat while true | until = repeat until true

10. BEST PRACTICES

- Always update condition variable
- Avoid infinite loops
- Use meaningful variable names
- Add break for safety
- Test your scripts carefully

11. QUICK REFERENCE

while	→ While condition is True
until	→ Until condition is True
do	→ Start of loop body
done	→ End of loop
break	→ Exit the loop
continue	→ Skip to next iteration

PRO TIP: Loops are powerful! Use them to save time and automate boring tasks.

Linux Shell Scripting

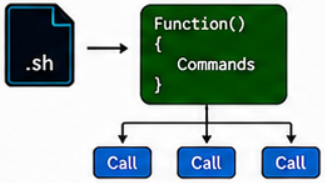
SHELL FUNCTIONS

Write Once, Use Many Times!

Terminal | Script | Functions | Reusable Code | Automation

1. WHAT IS A FUNCTION?

A function is a block of code that performs a specific task. It can be called (used) many times from anywhere in the script.



Functions help to organize code, reduce repetition and improve readability.

2. FUNCTION SYNTAX

```
name() {
    commands
    return [value]
}
```

Keyword	Meaning
name()	Function name
{	Start of function body
commands	Commands to be executed
return	Return value from function
}	End of function

Functions improve code reusability.

3. EXAMPLE: SIMPLE FUNCTION

Create a function and call it.

```
#!/bin/bash
# Function definition
greet() {
   echo "Hello, $1!"
   echo "Welcome to Shell Scripting."
   return 0
}

# Function call
greet "Linux User"
```

Output

```
Hello, Linux User!
Welcome to Shell Scripting.
```

4. FUNCTION WITH RETURN VALUE

Return a value from function.

```
#!/bin/bash
add() {
   local a=$1
   local b=$2
   local sum=$((a + b))
   return $sum
}

add 5 7
echo "Sum is: $?"
```

Output

```
Sum is: 12
```

5. FUNCTION WITH MULTIPLE ARGUMENTS

Pass multiple arguments to function.

```
#!/bin/bash
print_info() {
   echo "Name : $1"
   echo "Age  : $2"
   echo "City : $3"
}

print_info "Jeet" 32 "Amritsar"
```

Output

```
Name : Jeet
Age  : 32
City : Amritsar
```

6. FUNCTION WITHOUT ARGUMENTS

Function without any arguments.

```
#!/bin/bash
say_hello() {
   echo "Hello Everyone!"
   echo "Have a nice day."
}

say_hello
```

Output

```
Hello Everyone!
Have a nice day.
```

7. LOCAL VARIABLES IN FUNCTION

Use local variables to avoid conflicts.

```
#!/bin/bash
show() {
   local msg="This is a local variable"
   echo $msg
}
show
echo "Outside: $msg"
```

Output

```
This is a local variable
Outside:
(empty because msg is local)
```

8. FUNCTION CALLING ANOTHER FUNCTION

Functions can call other functions.

```
#!/bin/bash
hello() {
   echo "Hello!"
}
welcome() {
   hello
   echo "Welcome to Shell Script!"
}
welcome
```

Output

```
Hello!
Welcome to Shell Script!
```

9. CHECK IF FUNCTION EXISTS

Check if a function is defined.

```
#!/bin/bash
if declare -f myfunc > /dev/null; then
   echo "Function exists!"
   myfunc
else
   echo "Function does not exist!"
fi
```

Output (example)

```
Function exists!
```

10. BEST PRACTICES

- Use meaningful function names.
- Keep functions small and focused.
- Use local variables.
- Add comments to explain function.
- Return values for status/results.
- Reuse functions to reduce repetition.

CLEAN CODE
BETTER SCRIPTS
HAPPY YOU!

QUICK REFERENCE

function name() { ... }	Define function
name	Function name
$1, $2, ...	Arguments
return n	Return value
name arg1 arg2 ...	Call function
local var=value	Local variable

FUNCTION WORKFLOW

1. Define Function → 2. Call Function (Use it) → 3. Execute Commands → 4. Return (If any) → 5. Reuse Anywhere

PRO TIP

Break big tasks into small functions. It makes your script easy to read, test and maintain.

REMEMBER

Functions save time
Functions reduce code repetition
Functions make script professional

Use Functions. Write Smart Scripts. Automate Everything!

Linux
Shell Scripting

SCRIPT ARGUMENTS

Pass Data to Your Scripts from Command Line

Terminal | Script | Arguments | Command Line | Parameters

1. WHAT ARE SCRIPT ARGUMENTS?

When you run a shell script, you can pass values to it. These values are called arguments or parameters.

Example:

```
./script.sh  John  25  Amritsar
```

$0	$1	$2	$3
Script	1st Name	2nd Age	3rd City

Arguments help make scripts dynamic and reusable.

2. COMMON ARGUMENT VARIABLES

Variable	Meaning	Example
$0	Script name	./script.sh
$1	First argument	John
$2	Second argument	25
$3	Third argument	Amritsar
$@	All arguments (as separate strings)	"John" "25" "Amritsar"
$*	All arguments (as one string)	"John 25 Amritsar"
$#	Total number of arguments	3
$?	Exit status of last command	0 (success)

These variables are powerful in shell scripting.

3. EXAMPLE: DISPLAY ALL ARGUMENTS

Script: show_args.sh

```
#!/bin/bash
echo "Script Name    : $0"
echo "1st Argument   : $1"
echo "2nd Argument   : $2"
echo "3rd Argument   : $3"
echo "All Args (\$@)   : $@"
echo "All Args (\$*)   : $*"
echo "Total Args (\$#) : $#"
echo "Last Exit Code (\$?) : $?"
```

Run in Terminal:

```
./show_args.sh John 25 Amritsar
```

Output:

```
Script Name      : ./show_args.sh
1st Argument     : John
2nd Argument     : 25
3rd Argument     : Amritsar
All Args ($@)    : John 25 Amritsar
All Args ($*)    : John 25 Amritsar
Total Args ($#)  : 3
Last Exit Code ($?: 0
```

4. EXAMPLE: COUNT ARGUMENTS

Script: count_args.sh

```
#!/bin/bash
echo "You passed $# argument(s)."
echo "All arguments: $@"
```

Run in Terminal:

```
./count_args.sh one two three four
```

Output:

```
You passed 4 argument(s).
All arguments: one two three four
```

5. CHECK IF ARGUMENTS ARE PASSED

Script: check_args.sh

```
#!/bin/bash
if [ $# -eq 0 ]; then
    echo "No arguments provided!"
    echo "Usage: $0 <name> <age>"
    exit 1
fi
echo "Hello, $1! You are $2 years old."
```

Run without arguments:

```
./check_args.sh
```

Output:

```
No arguments provided!
Usage: ./check_args.sh <name> <age>
```

Run with arguments:

```
./check_args.sh Jeet 32
```

Output:

```
Hello, Jeet!
You are 32 years old.
```

6. SPECIAL ARGUMENT VARIABLES

$0	Name of the script
$1	First argument passed to script
$2	Second argument passed to script
$@	All arguments as separate strings
$*	All arguments as single string
$#	Total number of arguments
$?	Exit status of last command

7. PRACTICAL EXAMPLE: SUM NUMBERS

Script: sum.sh

```
#!/bin/bash
sum=0
for num in "$@"; do
    sum=$((sum + num))
done
echo "Sum of all numbers: $sum"
```

Run in Terminal:

```
./sum.sh 5 10 15 20
```

Output:

```
Sum of all numbers: 50
```

8. WHY USE SCRIPT ARGUMENTS?

- Make scripts dynamic and flexible
- Reuse scripts with different inputs
- Avoid hardcoding values
- Build powerful command line tools
- Accept files, numbers, names, options, etc.

Validate arguments in your script before using them for better error handling.

QUICK REFERENCE

$0	→ Script name
$1	→ First argument
$2	→ Second argument
$@	→ All arguments (separate)
$*	→ All arguments (single)
$#	→ Number of arguments
$?	→ Last command exit status

ARGUMENTS WORKFLOW

1. Write Script → 2. Add $1, $2.... in Script → 3. Run Script with Arguments → 4. Script Uses Arguments → 5. Get Results

NOTE

Always quote variables ("$@") to handle spaces in arguments.

EXAMPLE COMMANDS

```
./script.sh                 # Run without arguments
./script.sh one             # One argument
./script.sh a b c           # Multiple arguments
./script.sh "hello world"   # With spaces
./script.sh file.txt 10     # Mixed arguments
```

Linux Shell Scripting

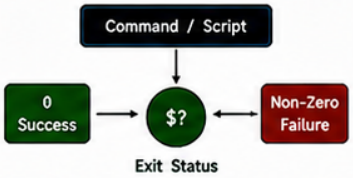
EXIT STATUS

Check Result of Last Command or Script

Terminal | Script | Exit Status | Status Codes | Success / Failure

1. WHAT IS EXIT STATUS?

Every command or script in Linux returns an exit status (also called status code).
You can check it using `$?`



By convention: `0` means success, any non-zero value means failure.

2. HOW TO CHECK EXIT STATUS

Syntax

```
$?
# Returns exit status of last command
```

Example

```
$ ls /home
$ echo $?
0    # Success

$ ls /no_such_dir
$ echo $?
2    # Failure (No such file or directory)
```

Always check exit status after the command you want to test.

3. COMMON EXIT STATUS CODES

Code	Meaning	Description
0	Success	Command completed successfully
1	General Error	General catch-all error
2	Misuse of shell	Invalid command or wrong usage
126	Permission denied	Command found but not executable
127	Command not found	Command not found
128	Invalid argument	Invalid argument to command
130	Script terminated	Script terminated by Ctrl+C
255	Exit status out of range	Exit status out of range

Different commands may have specific exit codes.
Use 'man <command>' to see details.

4. EXIT IN SHELL SCRIPTS

exit 0 (Success)

```
#!/bin/bash
echo "All done!"
exit 0
```

0 Success

Output

```
$ ./script.sh
echo $?
0
```

exit 1 (Failure)

```
#!/bin/bash
echo "Something went wrong!"
exit 1
```

1 Failure

Output

```
$ ./script.sh
echo $?
1
```

Use meaningful exit codes in your scripts to indicate success or failure.

5. PRACTICAL EXAMPLES

Example 1: Check Command

```
$ cp file.txt backup.txt
$ echo $?
0    # Success
```

Example 2: Check in Script

```
#!/bin/bash
cp file.txt backup.txt
if [ $? -eq 0 ]; then
    echo "File copied successfully!"
else
    echo "Failed to copy file."
fi
```

Always check $? immediately after the command or script.

6. USE EXIT STATUS IN CONDITIONS

```
#!/bin/bash
ls /tmp
if [ $? -eq 0 ]; then
    echo "Directory listed successfully."
else
    echo "Failed to list directory."
fi
```

Output (Example)

```
Directory listed successfully.
```

7. COMMON USE CASES

- Check if a command executed successfully
- Handle errors in scripts
- Run different actions based on success or failure
- Debug scripts easily
- Chain commands with && (success) and || (failure)

PRO TIP

Use 'set -e' at the top of your script to stop execution if any command fails.

8. EXIT STATUS SUMMARY

Exit Status	Meaning
0	Success (Everything is OK)
1	General Error / Failure
2	Incorrect Usage / Misuse of shell
126	Permission denied
127	Command not found
>128	Terminated by signal (e.g., Ctrl+C)

QUICK REFERENCE

```
$?          -> Exit status of last command
exit 0      -> Success
exit 1      -> Failure
0           -> Success code
Non-zero    -> Failure code
```

EXIT STATUS WORKFLOW

1. Run Command or Script
2. Check with $?
3. Evaluate Status (0 or Non-zero)
4. Take Action (Success / Failure)

BEST PRACTICES

- Always check exit status
- Use meaningful exit codes
- Handle errors gracefully
- Keep scripts reliable and user-friendly

REMEMBER

Good scripts don't just run commands, they also check if commands succeed or fail!

Check $?. Know the Status. Write Better Scripts!

Linux Shell Scripting

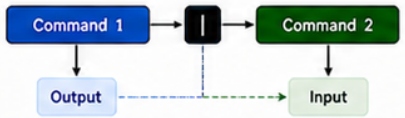
PIPES (|)

Connect Commands. Pass Output as Input.

Terminal | Script | Pipes | Filter | Process

1. WHAT IS A PIPE?

A pipe (|) is used to send the output of one command (Command 1) as the input to another command (Command 2).
It creates a pipeline between commands.

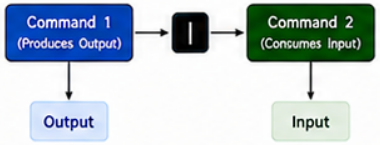


Output of Command 1 ----> becomes Input of Command 2 ---->

Pipes allow you to chain commands together to filter, process and transform data.

2. SYNTAX

```
Command 1 | Command 2
```



Symbol	Name	Meaning
|	Pipe	Send output of left command as input to right command

Pipes do not create intermediate files. They work in memory, so it's fast!

3. EXAMPLES

No.	Command	Explanation
1	`ls -l | less`	List files in long format and view page by page
2	`ps aux | grep ssh`	Show only lines containing 'ssh' from process list
3	`cat file.txt | grep error`	Show lines with 'error' from file.txt
4	`df -h | sort -h`	Sort disk usage by size (human readable)
5	`netstat -tuln | grep LISTEN`	Show only listening ports
6	`who | wc -l`	Count number of logged-in users

The output of the left command becomes the input of the right command.

4. REAL TIME EXAMPLES

Example 1: List and view one screen at a time

```
$ ls -l | less
```

Output (less)

```
-rw-r--r--   1 user user    1234  file.txt
-drwx-xr-x   2 user user    4096  folder/
-rw-r--r--   1 user user     234  note.txt
...
```

Example 2: Find ssh processes

```
$ ps aux | grep ssh
user      1234   0.0  0.1  22400  1236 ?      Ss   10:00   0:00 sshd
user      5678   0.0  0.1   9880   988 pts/0  S+   10:05   0:00 grepssh
```

Example 3: Count users

```
$ who | wc -l
```

Output

```
2
```

5. PIPELINE WITH MULTIPLE COMMANDS

You can chain multiple commands using multiple pipes.

```
Command 1 | Command 2 | Command 3 | ... | Command N
```

Example:

```
$ cat access.log | grep "GET" | awk '{print $1}' | sort | uniq -c
  | sort -nr | head -10
```

Each command works on the output of the previous one.

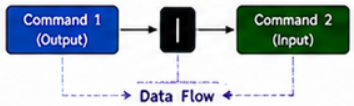
6. COMMON USE CASES

Filtering	Show only important lines	ps aux | grep nginx
Sorting	Sort output by size or name	ls -l | sort -k5 -n
Counting	Count lines, words or chars	cat file.txt | wc -l
Searching	Search text in output	dmesg | grep error
Processing	Extract and format data	df -h | awk '{print $5}'

Pipes help in building powerful one-liners without creating temporary files.

7. HOW PIPES WORK?

1. Command 1 produces output.
2. The output is sent to the pipe (|).
3. Command 2 receives that output as input.
4. Command 2 processes the input and shows the result.



8. IMPORTANT NOTES

- Pipes work left to right.
- Use quotes to protect special characters.
- Errors go to screen (unless redirected).
- Pipes do not store output in files.
- Very useful in automation and one-liners.

PRO TIP

Mastering pipes will make you faster and smarter in terminal.
Use pipes + filters + tools = Power! 💪

QUICK REFERENCE

```
|          → Pipe (output to input)
Command1 | Command2  → Chain commands
cmd1 | cmd2 | cmd3   → Multiple pipelines
Standard Input       → Data coming in
Standard Output      → Data going out
Standard Error       → Error messages
```

PIPE WORKFLOW

1. Run Command 1 (Produce Output)
2. Send Output through Pipe
3. Command 2 Processes Input
4. Transform / Filter Data
5. Show Final Result

BEST PRACTICES

- Use pipes to combine simple tools.
- Keep commands simple and readable.
- Test each part of pipeline separately.
- Use `man` command to learn more.

REMEMBER

Pipes are the backbone of Linux power.
Learn once, use forever!

Think: Output → Input → Result 💯

Pipe it Right, Get it Fast! Automate, Filter, Process! 🚀

Linux Shell Scripting

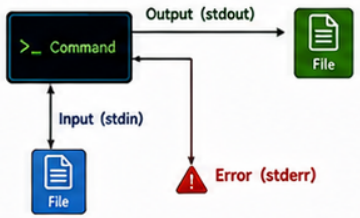
REDIRECTION

Control Input and Output of Commands

Terminal | Script | Redirection | stdin | stdout | stderr

1. WHAT IS REDIRECTION?

Redirection is used to send the output of a command to a file or use a file as input for a command.



Redirection helps to save output, read input from files and handle errors.

2. REDIRECTION OPERATORS

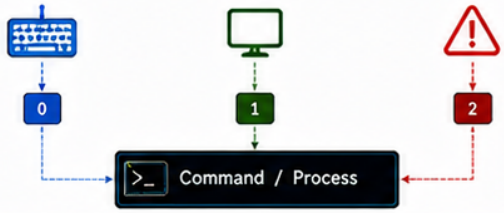
Operator	Name	Description	Example	Meaning
>	Redirect Output	Redirect stdout to a file. (Overwrite file if exists)	command > file.txt	Save output to file.txt (Overwrite)
>>	Append Output	Redirect stdout and append to a file.	command >> file.txt	Add output to file.txt (Append)
<	Redirect Input	Take input from a file instead of keyboard.	command < input.txt	Use input from file
2>	Redirect Error	Redirect stderr to a file. (Overwrite)	command 2> error.txt	Save errors to error.txt (Overwrite)
2>>	Append Error	Append stderr to a file.	command 2>> error.txt	Add errors to error.txt (Append)
> file 2>&1	Redirect Both	Redirect stdout and stderr to same file.	command > out.txt 2>&1	Save output and errors to out.txt
&> file	Redirect All (Shortcut)	Redirect stdout and stderr to same file. (Bash)	command &> all.log	Save both output and errors to all.log

3. STANDARD STREAMS

stdin (0) — Standard Input — Input to the command

stdout (1) — Standard Output — Normal output of the command

stderr (2) — Standard Error — Error output of the command



By default: 0 = stdin, 1 = stdout, 2 = stderr

4. EXAMPLES

A) Redirect Output (Overwrite)

```
$ ls -l > files.txt
$ cat files.txt

total 12
-rw-r--r-- 1 user user  123 file1.txt
-rw-r--r-- 1 user user  456 file2.txt
```

Saves the output of 'ls -l' to files.txt (Old content will be lost)

B) Append Output

```
$ echo "Hello" >> data.txt
$ echo "World" >> data.txt
$ cat data.txt

Hello
World
```

Appends output to data.txt (Old content will be kept)

C) Redirect Input

```
$ cat input.txt
Linux is awesome!
Shell scripting is powerful.

$ wc -l < input.txt
2
```

Takes input from input.txt instead of keyboard

D) Redirect Error (Overwrite)

```
$ ls no_such_file 2> error.txt
$ cat error.txt

ls: cannot access 'no_such_file':
No such file or directory
```

Saves error output to error.txt (Overwrite if exists)

E) Append Error

```
$ ls no_file 2>> error.log
$ grep not_found file.txt 2>> error.log
$ cat error.log

ls: cannot access 'no_file':
No such file or directory
grep: file.txt: No such file or directory
```

Appends errors to error.log

F) Redirect Output and Error (Separate Files)

```
$ command > out.txt 2> err.txt
```

out.txt (Output) | err.txt (Error)

Output goes to out.txt
Errors go to err.txt

G) Redirect Both to Same File

```
$ command > all.txt 2>&1
# or
$ command &> all.txt
```

all.txt (Output + Error)

Both output and errors go to the same file

5. QUICK REFERENCE

>	file	→	Redirect stdout (overwrite)
>>	file	→	Redirect stdout (append)
<	file	→	Redirect stdin
2>	file	→	Redirect stderr (overwrite)
2>>	file	→	Redirect stderr (append)
> file 2>&1		→	Redirect both stdout & stderr
&>	file	→	Redirect both (Bash shortcut)

6. REDIRECTION WORKFLOW

1. Run Command → 2. Apply Redirection → 3. Send Output / Take Input / Handle Error → 4. Save to File / Use as Input

7. PRACTICAL USE CASES

- Save command output to log files
- Create reports automatically
- Handle errors in scripts
- Feed input from files to commands
- Keep terminal clean and organized

8. BEST PRACTICES

- Always quote filenames
- Use >> for logs (don't overwrite)
- Separate error and output when needed
- Check log files for debugging
- Use &> for simple combine

9. REMEMBER

Redirection gives you power over input, output and errors. Use it smartly!

PRO TIP: Master redirection to write powerful, clean and professional shell scripts!

Linux Shell Scripting

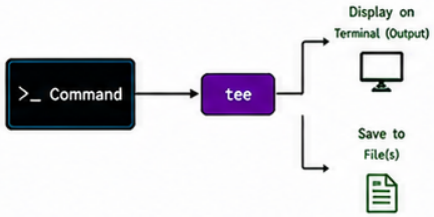
TEE COMMAND `tee`

Read from standard input and write to both output and files

Terminal | Script | tee | Pipeline | File | Save | Display

1. WHAT IS TEE?

The tee command reads from standard input and writes the data to both standard output (terminal) and one or more files.



tee is very useful when you want to see the output in the terminal and also save it to a file.

2. SYNTAX

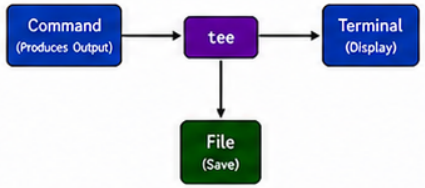
```
command | tee [options] file(s)
```

Part	Meaning
command	The command whose output you want to capture
|	Pipe: sends output of command to tee
tee	Reads input and writes to output and file(s)
[options]	Optional flags (like -a)
file(s)	File name(s) to save output

tee works with pipelines.
It does not replace the output.

3. HOW TEE WORKS?

`command | tee file`



- Reads input from the left (pipeline).
- Writes the same input to terminal (stdout).
- Also writes the same input to file.
- Both output and file content are same.

4. EXAMPLES

#	Example	Explanation
1	`ls -l | tee list.txt`	• List files and directories. • Show output in terminal and save it to list.txt
2	`echo "Hello Linux" | tee hello.txt`	• Print text and save it to hello.txt
3	`dmesg | tee dmesg.log`	• Show kernel messages and save them to dmesg.log
4	`cat access.log | tee log.txt | grep "GET"`	• Save output to log.txt and also filter it with grep
5	`ip a | tee -a network.log`	• Append output to network.log (won't overwrite)

5. COMMON OPTIONS

Option	Meaning	Example
-a	Append to the file (do not overwrite)	command | tee -a file.txt
--help	Show help	tee --help
--version	Show version	tee --version

By default, tee overwrites the file.
Use -a to append.

6. PRACTICAL EXAMPLES

1. Save command output

```
$ df -h | tee disk_usage.txt
```

Output (Terminal):

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   20G   28G  42% /
tmpfs            2G    1M    2G   1% /dev/shm
...
```

File: disk_usage.txt
(Same content saved)

2. Append output to file

```
$ echo "First line" | tee -a notes.txt
$ echo "Second line" | tee -a notes.txt
```

File: notes.txt

```
First line
Second line
```

3. Use in pipeline

```
$ cat access.log | tee all.log | grep "404"
```

Terminal (shows filtered output):

```
192.168.1.1 - - [20/May/2025:10:10:10] "GET /404" 404
192.168.1.2 - - [20/May/2025:10:11:20] "GET /404" 404
```

File: all.log
(Contains full access.log data)

7. USE CASES

- Save command output to a file while seeing it in terminal.
- Create logs of scripts or system info.
- Useful in debugging and monitoring.
- Append data to log files.
- Works great in automation and pipelines.

PRO TIP

Use tee to keep both a live view and a saved record.
Perfect for logs and reports!

QUICK REFERENCE

```
command | tee file          → Output + Save
command | tee -a file       → Append
command | tee file1 file2   → Save to multiple files
command | tee /dev/null     → Output to terminal only
```

TEE WORKFLOW

1. Run Command (Generate Output)
2. Pass to tee via Pipeline
3. Display on Terminal
4. Save to File(s)

BEST PRACTICES

- Use meaningful filenames.
- Use -a to append, not overwrite.
- Combine tee with grep, awk, sort, etc.
- Keep logs organized in folders.

REMEMBER

tee = Terminal + File
See it here, Save it there!
Display and Save together like a pro!

Use tee Command. See Output. Save Output. Stay Smart!

Linux Shell Scripting

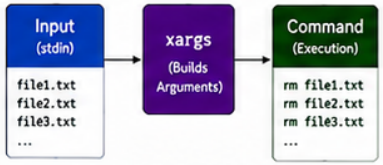
XARGS COMMAND `xargs`

Build and Execute Commands from Standard Input

Terminal | Script | xargs | Pipeline | Files | Automation

1. WHAT IS XARGS?

xargs reads input from stdin (standard input) and converts it into arguments for a command. It helps to build and execute commands efficiently. Great for automation and batch processing.



It takes output from one command and passes it as arguments to another command.

2. SYNTAX

```
command | xargs [options] [command] [initial-arguments]
```

Part	Meaning
command	The command that produces output (stdin)
xargs	Reads input and constructs arguments
[options]	Modify how xargs runs the command
[command]	The command to execute with arguments
[initial-arguments]	Optional arguments to pass before stdin items

Without a command, xargs echoes the arguments.
Example: `echo "a b" | xargs` → a b

3. HOW XARGS WORKS?

producer command | xargs [command]

- Reads input from stdin.
- Splits input into arguments (by default, by whitespace).
- Executes the command with those arguments.
- Can run multiple arguments in batches.

4. EXAMPLES

#	Example	Explanation
1	`ls | xargs rm`	Delete files listed by ls
2	`find . -name "*.log" | xargs rm`	Find .log files and delete them
3	`cat file.txt | xargs -n 1 echo`	Print each line using echo
4	`echo "a b c d" | xargs echo`	Output: a b c d
5	`ls | xargs -I {} mv {} backup/`	Move files to backup/ using {} as a placeholder
6	`find . -type f | xargs wc -l`	Count lines in all files

xargs is very powerful for batch processing and automation.

5. COMMON OPTIONS

Option	Meaning	Example
-n N	Use at most N arguments per command	xargs -n 5 cmd
-I {}	Replace {} with input lines	xargs -I {} rm {}
-d DELIM	Use DELIM as input delimiter	xargs -d '\n' cmd
-0	Input is null-terminated (for -print0)	xargs -0 cmd
-P N	Run up to N commands in parallel	xargs -P 4 cmd
-t	Print the command before executing	xargs -t cmd
-r	Do not run command if input is empty	xargs -r cmd

6. PRACTICAL EXAMPLES

A) Delete .tmp files

```
find . -name "*.tmp" | xargs rm -f
```

Find all .tmp files and delete them.

B) Count lines in files

```
find . -type f -name "*.txt" | xargs wc -l
```

Count total lines in all .txt files.

C) Rename with mv

```
ls *.txt | xargs -I {} mv {} {}.bak
```

Rename all .txt files to .bak extension.

D) Compress files

```
find . -name "*.log" | xargs gzip
```

Compress all .log files using gzip.

E) Use with specific args

```
echo "one two three" | xargs -n 1 echo
```

Run echo once for each word.

F) Parallel execution

```
find . -type f -name "*.jgg" | \
  xargs -P 4 -I {} convert {} {}.png
```

Convert images in parallel using 4 jobs.

7. USE CASES

- Delete, move or rename multiple files
- Process large lists of files or data
- Automate repetitive tasks
- Work with find, grep, ls and more
- Speed up tasks using parallel execution

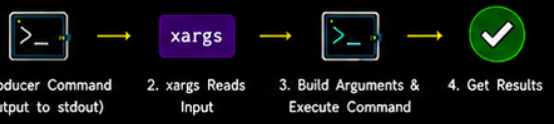
PRO TIP

Use find -print0 | xargs -0 to safely handle filenames with spaces or special characters.

QUICK REFERENCE

```
command | xargs cmd          → Pass input as args
command | xargs -n N cmd     → Max N args per run
command | xargs -I {} cmd    → Replace {} with line
command | xargs -0 cmd       → Null-terminated input
command | xargs -P N cmd     → Run N in parallel
command | xargs -t cmd       → Show command first
command | xargs -r cmd       → Skip if no input
```

XARGS WORKFLOW



BEST PRACTICES

- Always quote variables in commands.
- Use -print0 with xargs -0 for safety.
- Test with -t before actual execution.
- Use -P for faster processing.

REMEMBER

xargs turns input into action!
Less typing, more power.
Automate smartly, work efficiently!

xargs Makes Big Jobs Easy – Pipe, Pass, Process, and Power Up!

Linux Shell Scripting

CRON JOBS

Automate Tasks. Save Time. Let Cron Work for You!

cron | crontab -e | crontab -l | Schedule | Job | Automation | Time

1. WHAT IS CRON?

Cron is a time-based job scheduler in Linux.
It runs commands or scripts automatically
at specified time and date.
It is perfect for automation and routine tasks.

Time (Schedule) → cron (Scheduler) → Job / Script (Command)

Cron runs in the background and does not require user login.

2. CRONTAB COMMANDS

Command	Description
`crontab -e`	Edit your crontab file.
`crontab -l`	List your current cron jobs.
`crontab -r`	Remove all your cron jobs.
`crontab -e -u username`	Edit crontab for another user (root only).
`crontab -l -u username`	List crontab for another user.

Crontab files are stored in /var/spool/cron/crontabs/
One file per user.

3. CRON SYNTAX (SCHEDULE FORMAT)

```
* * * * * command_to_execute
```

Min	Hour	Day of Month	Month	Day of Week
0-59	0-23	1-31	1-12	0-7 (0 or 7 = Sun)

Special Characters

- `*` → Any value
- `,` → List (e.g., 1,3,5)
- `-` → Range (e.g., 1-5)
- `/` → Step values (e.g., */15)

Examples

- `* * * * *` Every minute
- `0 * * * *` Every hour
- `0 0 * * *` Every day at midnight
- `0 0 * * 0` Every Sunday at midnight

4. EXAMPLES OF CRON JOBS

Schedule	Description	Example Command
`* * * * *`	Every minute	`* * * * * /path/script.sh`
`0 * * * *`	Every hour	`0 * * * * /path/script.sh`
`0 0 * * *`	Every day at midnight	`0 0 * * * /path/script.sh`
`30 2 * * *`	Every day at 02:30 AM	`30 2 * * * /path/script.sh`
`0 0 * * 0`	Every Sunday at midnight	`0 0 * * 0 /path/script.sh`
`*/15 * * * *`	Every 15 minutes	`*/15 * * * * /path/script.sh`
`0 9 1 * *`	At 09:00 AM on 1st of every month	`0 9 1 * * /path/script.sh`

Use absolute path in commands (e.g., /usr/bin/php)

5. HOW TO CREATE A CRON JOB

1. Type: `crontab -e`
2. Add your job in the format:
   ```
   * * * * * /path/to/command
   ```
3. Save and exit the editor.

```
$ crontab -e
no crontab for user.  Use -u to specify user.
Adding cron job...
$
```

Editor will open (vi by default). Add your job and save.

6. VIEW AND MANAGE CRON JOBS

List your cron jobs:

```
$ crontab -l
# m h dom mon dow command
0 0 * * *  /home/user/backup.sh
30 2 * * *  /usr/bin/logrotate /etc/logrotate.conf
$
```

Edit another user (root):

```
# crontab -e -u username
# crontab -l -u username
```

Remove all cron jobs:

```
$ crontab -r
$
```

Important Notes

- Cron jobs run in a minimal environment.
- Set required PATH in your script if needed.
- Output is mailed to the user (check mail) or redirected to a log file.

7. OUTPUT & LOGGING

By default, cron sends output to the mail of the user.
It is better to redirect output to a log file.

Example with logging:

```
0 1 * * * /path/script.sh >> /var/log/script.log 2>&1
```

Cron Job (Runs) → Command / Script (Executes) → (Output Saved)

2>&1 → Redirects both standard output (1) and standard error (2) to the log file.

8. COMMON USE CASES

- Daily Backup
- Log Rotation
- Cleanup Temporary Files
- Send Reports / Emails
- Database Maintenance
- System Monitoring Scripts
- Automate Repetitive Tasks

PRO TIP

Always test your script manually before adding it to cron.

QUICK REFERENCE

`crontab -e`	→ Edit your cron jobs
`crontab -l`	→ List your cron jobs
`crontab -r`	→ Remove all cron jobs
`crontab -e -u user`	→ Edit another user
`crontab -l -u user`	→ List another user

CRON WORKFLOW

1. Set Schedule (Time) → 2. Cron Scheduler Triggers (cron) → 3. Run Command (Job) → 4. Output / Log / Email

BEST PRACTICES

- Keep scripts simple and reliable.
- Use absolute paths.
- Log all outputs and errors.
- Test before scheduling.
- Avoid heavy tasks on system load.

REMEMBER

Cron is a powerful tool for automation.
Schedule once,
and let your system do the rest!

Work Smarter, Not Harder — Automate with Cron!

Linux Shell Scripting

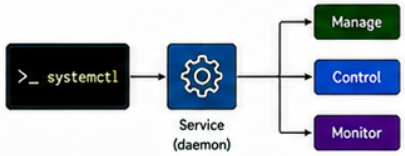
SYSTEMCTL COMMAND

Manage and Control systemd Services Easily

systemctl | start | stop | restart | status | enable | disable | Service

1. WHAT IS SYSTEMCTL?

systemctl is a command-line utility used to control the systemd system and service manager. It can start, stop, restart, enable, disable and check status of services (daemons) on Linux.



It works with systemd, the default init system in most modern Linux distributions.

2. COMMON SYSTEMCTL COMMANDS

Command	Description	Example
start	Start a service immediately.	sudo systemctl start nginx
stop	Stop a running service.	sudo systemctl stop nginx
restart	Restart a service.	sudo systemctl restart nginx
status	Show the status of a service.	sudo systemctl status nginx
enable	Enable a service to start on boot.	sudo systemctl enable nginx
disable	Disable a service from starting on boot.	sudo systemctl disable nginx

3. SERVICE STATES

State	Meaning
active (running)	Service is running successfully.
active (exited)	Service ran and exited successfully.
failed	Service failed to start.
inactive (dead)	Service is not running.
activating	Service is starting...
deactivating	Service is stopping...

4. EXAMPLES

Check status of SSH service

```
$ systemctl status sshd
```

Start Apache service

```
$ sudo systemctl start apache2
```

Restart Docker service

```
$ sudo systemctl restart docker
```

Enable Nginx on boot

```
$ sudo systemctl enable nginx
```

Disable Nginx from boot

```
$ sudo systemctl disable nginx
```

Use sudo with systemctl for most commands.

5. SERVICE STATUS EXAMPLE

```
$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2025-05-20 10:15:30 IST; 2h 13min ago
     Docs: man:nginx(8)
 Main PID: 1234 (nginx)
    Tasks: 3 (limit: 2316)
   Memory: 4.2M
   CGroup: /system.slice/nginx.service
           ├─1234 nginx: master process /usr/sbin/nginx -g daemon on;
           ├─1235 nginx: worker process
           └─1236 nginx: worker process
```

- active (running) : Service is running
- active (exited) : Service ran and exited
- failed : Service failed
- inactive (dead) : Service is not running

6. LIST SERVICES

List all loaded services:

```
$ systemctl list-units --type=service --state=all
```

List only running services:

```
$ systemctl list-units --type=service --state=running
```

7. CHECK IF SERVICE IS ENABLED

Check if a service is enabled to start on boot:

```
$ systemctl is-enabled nginx
# Output: enabled
```

8. RELOAD SYSTEMD DAEMON

After making changes to service files, reload systemd daemon:

```
$ sudo systemctl daemon-reload
```

9. MASK AND UNMASK SERVICE

Mask a service (completely disable it):

```
$ sudo systemctl mask nginx
```

Unmask a service:

```
$ sudo systemctl unmask nginx
```

10. COMMON USE CASES

- Start / Stop / Restart services
- Check service status and logs
- Enable services to run on boot
- Disable services from boot
- Manage system services easily
- Automate tasks with systemd services

PRO TIP

systemctl works with systemd units like services, sockets, timers, mounts, targets and more.
Use tab key for auto-completion.

QUICK REFERENCE

```
systemctl start <service>    → Start a service
systemctl stop <service>     → Stop a service
systemctl restart <service>  → Restart a service
systemctl status <service>   → Check status
systemctl enable <service>   → Enable on boot
systemctl disable <service>  → Disable from boot
```

SYSTEMCTL WORKFLOW

1. Check Status (systemctl status) → 2. Start (systemctl start) → 3. Stop (systemctl stop) → 4. Restart (systemctl restart) → 5. Enable (systemctl enable) → 6. Verify Status (systemctl status)

BEST PRACTICES

- Always check status after changes.
- Use enable for persistent services.
- Read logs with: journalctl -u <service>
- Keep services minimal and secure.

REMEMBER

systemctl is your control center for systemd services.
Master it and automate everything!

Control Services Like a Pro with systemctl!

Linux Shell Scripting

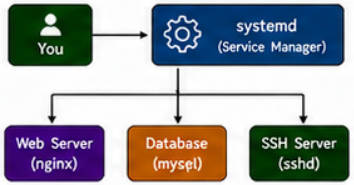
SERVICES & LOGS

Manage Services and Monitor Logs Like a Pro

systemctl | journalctl | service | Logs | Errors | Monitoring | Debugging

1. WHAT ARE SERVICES?

Services are background programs (daemons) that run on your system to perform important tasks.
systemd is the default service manager in most modern Linux distributions.



Services start on boot, run in background, and can be managed with systemctl.

2. COMMON systemctl COMMANDS

Command	Description	Example
start	Start a service	sudo systemctl start nginx
stop	Stop a running service	sudo systemctl stop nginx
restart	Restart a service	sudo systemctl restart nginx
status	Show status of a service	sudo systemctl status nginx
enable	Enable a service to start on boot	sudo systemctl enable nginx
disable	Disable a service from starting on boot	sudo systemctl disable nginx
reload	Reload configuration without stopping service	sudo systemctl reload nginx

3. CHECK SERVICE STATUS

```
$ systemctl status nginx
● nginx.service - A high performance web server
  Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
  Active: active (running) since Mon 2025-05-20 10:15:30 IST; 2h 15min ago
    Docs: man:nginx(8)
Main PID: 1234 (nginx)
   Tasks: 3 (limit: 2316)
  Memory: 4.2M
     CPU: 0.123s
  CGroup: /system.slice/nginx.service
          ├─1234 nginx: master process /usr/sbin/nginx -g daemon on;
          ├─1235 nginx: worker process
          └─1236 nginx: worker process
```

- active (running) : Service is running
- active (exited) : Service ran and exited successfully
- failed : Service failed to start
- inactive (dead) : Service is not running

4. VIEW LOGS WITH journalctl

journalctl is used to view and analyze logs from systemd and services.

Command	Description
journalctl	Show all logs
journalctl -xe	Show logs with errors (most recent first)
journalctl -u nginx	Show logs of a specific service
journalctl -u nginx -f	Follow (live) logs of a service
journalctl -b	Show logs from current boot
journalctl --since "1 hour ago"	Show logs from last 1 hour
journalctl --since today	Show logs from today
journalctl -p err	Show only error logs

Great for monitoring, troubleshooting and debugging.

5. EXAMPLE: VIEW SERVICE LOGS

```
$ journalctl -u nginx --since "1 hour ago"
May 20 09:15:30 server systemd[1]: Starting nginx.service - A high performance web server...
May 20 09:15:30 server nginx[1234]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
May 20 09:15:30 server nginx[1234]: nginx: configuration file /etc/nginx/nginx.conf test is successful
May 20 09:15:31 server systemd[1]: Started nginx.service - A high performance web server.
May 20 09:45:12 server nginx[1235]: 192.168.1.10 - - [20/May/2025:09:45:12 +0530] "GET / HTTP/1.1" 200 612
May 20 09:48:25 server nginx[1235]: 192.168.1.11 - - [20/May/2025:09:48:25 +0530] "GET /about.html" 404 555
May 20 09:50:40 server nginx[1235]: 192.168.1.12 - - [20/May/2025:09:50:40 +0530] "POST /login" 500 1234
```

Logs help you understand what happened, when, and why.

6. COMMON SERVICES

Service	Description	Default Port
nginx	Web Server	80, 443
ssh	Secure Shell	22
mysql	Database Server	3306
apache2	Web Server	80, 443
cron	Task Scheduler	-
docker	Container Engine	-

7. MONITORING & DEBUGGING TIPS

- Use systemctl status to quickly check service health.
- Use journalctl -u <service> to see service specific logs.
- Use journalctl -xe to find recent errors.
- Logs are stored in /var/log (traditional logs).
- Keep services enabled only if needed.
- Regularly review logs for errors and warnings.

ERROR EXAMPLE (journalctl -xe)

```
$ journalctl -xe
May 20 10:00:01 server nginx[1234]: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
May 20 10:00:01 server systemd[1]: nginx.service: Main process exited, code=exited, status=1/FAILURE
May 20 10:00:01 server systemd[1]: nginx.service: Failed with result 'exit-code'.
```

8. USE CASES

- Start web server: systemctl start nginx
- Auto start on boot: systemctl enable nginx
- Check if running: systemctl status nginx
- View errors: journalctl -u nginx -xe
- Follow live logs: journalctl -u nginx -f
- Debug issues and fix faster
- Monitor system and services 24/7

PRO TIP
Combine systemctl and journalctl for powerful service management and deep log analysis!

QUICK REFERENCE

```
systemctl start <service>     → Start service
systemctl stop <service>      → Stop service
systemctl restart <service>   → Restart service
systemctl status <service>    → Check status
systemctl enable <service>    → Enable on boot
systemctl disable <service>   → Disable on boot
journalctl -u <service>       → View service logs
journalctl -xe                → View recent errors
```

SERVICE MANAGEMENT WORKFLOW

1. Manage Service (Start / Stop / Restart) → 2. Check Status (systemctl status) → 3. View Logs (journalctl -u) → 4. Analyze Issues (Find Errors) → 5. Fix & Monitor (Keep Healthy)

BEST PRACTICES

- Keep services updated and secure.
- Review logs daily.
- Automate log rotation.
- Monitor critical services.
- Act on errors quickly.

REMEMBER

Services keep your system alive.
Logs tell you the real story.
Monitor, Analyze, and Optimize like a Pro!

Manage Services. Read Logs. Fix Problems. Keep Your System Running Smooth!

Linux Shell Scripting

PACKAGE MANAGEMENT

Install, Update, Remove and Manage Packages Easily

apt | apt update | apt install | apt remove | dnf / yum | pacman | Packages

1. WHAT IS PACKAGE MANAGEMENT?

Package management tools help you install, update, configure and remove software packages on your Linux system.
Different distributions use different tools.

Keep your system updated for security, performance and stability.

2. DEBIAN / UBUNTU – APT

Command	Description
sudo apt update	Update package lists from repositories
sudo apt upgrade	Upgrade installed packages
sudo apt install <pkg>	Install a new package
sudo apt remove <pkg>	Remove a package
sudo apt purge <pkg>	Remove with config files
sudo apt autoremove	Remove unused dependencies
sudo apt search <pkg>	Search for packages
sudo apt show <pkg>	Show package information
sudo apt list --installed	List installed packages

```
$ sudo apt update
$ sudo apt install vim curl git
```

3. RHEL / FEDORA – DNF / YUM

Command	Description
sudo dnf check-update	Check for available updates
sudo dnf upgrade	Upgrade installed packages
sudo dnf install <pkg>	Install a new package
sudo dnf remove <pkg>	Remove a package
sudo dnf info <pkg>	Show package information
sudo dnf search <pkg>	Search for packages
sudo dnf list installed	List installed packages
sudo dnf clean all	Clean cached packages

```
$ sudo dnf check-update
$ sudo dnf install httpd wget
```

On older systems (CentOS/RHEL 7), use yum instead of dnf.

4. ARCH LINUX – PACMAN

Command	Description
sudo pacman -Syu	Synchronize and update system
sudo pacman -S <pkg>	Install a new package
sudo pacman -R <pkg>	Remove a package
sudo pacman -Rs <pkg>	Remove with dependencies
sudo pacman -Qs <pkg>	Search for a package
sudo pacman -Qi <pkg>	Show package information
sudo pacman -Q	List installed packages
sudo pacman -Sc	Clean package cache

```
$ sudo pacman -Syu
$ sudo pacman -S neofetch
```

Pacman is fast and lightweight.
Great for Arch based systems.

5. COMMON OPTIONS (APT / DNF / PACMAN)

Option	Meaning
-y	Assume yes to all prompts
-q	Quiet mode (less output)
--help	Show help
--version	Show version
--reinstall	Reinstall a package
--no-cache	Do not use cache
--allowerasing	Allow erasing (DNF specific)
--needed	Do not reinstall if already latest

Example:

```
sudo apt install -y nginx
sudo dnf install -y nginx
sudo pacman -S --noconfirm nginx
```

6. LIST INSTALLED PACKAGES

APT (Debian/Ubuntu)

```
$ dpkg --list
```

DNF/YUM (RHEL/Fedora)

```
$ dnf list installed
```

PACMAN (Arch Linux)

```
$ pacman -Q
```

Know what's installed on your system.

7. UPDATE SYSTEM

Debian / Ubuntu (APT)

```
$ sudo apt update && sudo apt upgrade -y
```

RHEL / Fedora (DNF/YUM)

```
$ sudo dnf upgrade --refresh -y
```

Arch Linux (Pacman)

```
$ sudo pacman -Syu
```

Regular updates keep your system safe and stable.

8. REMOVE UNUSED PACKAGES

APT (Debian/Ubuntu)

```
$ sudo apt autoremove -y
```

DNF (RHEL/Fedora)

```
$ sudo dnf autoremove -y
```

PACMAN (Arch Linux)

```
$ sudo pacman -Rns $(pacman -Qtdq)
```

Clean your system and free up disk space.

9. FIND PACKAGE INFORMATION

APT

```
$ apt show nginx
```

DNF

```
$ dnf info nginx
```

PACMAN

```
$ pacman -Qi nginx
```

Get detailed info about any package.

10. USE CASES

- Install servers: nginx, apache2, mysql
- Install dev tools: git, vim, curl, python3
- Remove unwanted software
- Update system regularly
- Check for security updates
- Keep system clean and optimized

PRO TIP

Always update before installing new packages.
Read package info before installing.
Use official repositories for safety.

QUICK REFERENCE

apt update	→ Update package lists
apt install pkg	→ Install a package
apt remove pkg	→ Remove a package
dnf install pkg	→ Install (RHEL/Fedora)
pacman -S pkg	→ Install (Arch Linux)

PACKAGE MANAGEMENT WORKFLOW

1. Update List (Get Latest Info) → 2. Search Package (Find What You Need) → 3. Install Package (Add to System) → 4. Manage / Configure (Use & Configure) → 5. Remove Unused (Keep Clean)

BEST PRACTICES

- Keep system and packages updated.
- Use official repositories.
- Remove unused packages and caches.
- Backup important configs.
- Read documentation when needed.

REMEMBER

Good package management = Secure System + Better Performance + Happy User!

Install Smart. Update Regularly. Remove Unwanted. Keep Your Linux System Healthy!